ALGORITMOS E LÓGICA DE PROGRAMAÇÃO

2ª edição revista e ampliada

Com linguagem simples e didática – sem, no entanto, fugir da complexidade do assunto –, o livro procura tornar a lógica de programação prática, além de mostrar aos estudantes um caminho mais adequado na construção dos algoritmos. A abstração de procedimentos e dados é um dos maiores problemas para os estudantes nos cursos introdutórios, e, para tentar escapar das dificuldades, os autores utilizam uma arquitetura de computador simples, baseada na arquitetura de Von Neumann, de maneira a fixar os conceitos relacionados à operação de computadores. Um dos principais objetivos do livro é fazer que o estudante consiga no futuro relacionar os aspectos abstratos da computação com sua implementação, e ainda incentivar a necessidade de escrever os algoritmos antes de sua implementação propriamente dita.

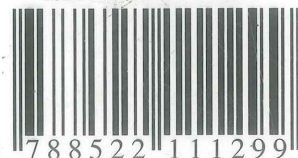
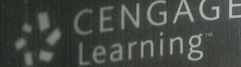
A descrição dos algoritmos no texto é mostrada por meio de fluxogramas. Também são apresentadas mais duas formas de representação de algoritmos: diagramas de Nassi-Schneidermann e o pseudocódigo Portugol, que emprega descrição textual e estruturada da solução de um problema. São apresentadas três maneiras de representação de algoritmos, tornando a fixação dos conceitos ainda mais fácil. A obra traz figuras, ilustrações e fotografias que enriquecem o conteúdo.

Nesta 2ª edição revista e ampliada foram acrescentadas resoluções de alguns dos exercícios propostos que complementam o texto e o aprendizado.

APLICAÇÕES

Livro destinado a disciplinas introdutórias de lógica de programação ministradas em diversos cursos.

Para suas soluções de curso e aprendizado, visite **www.cengage.com.br**

ISBN 13: 978-85-221-1129-9
ISBN 10: 85-221-1129-4

ALGORITMOS E LÓGICA DE PROGRAMAÇÃO

2ª edição revista e ampliada

MARCO A. FURLAN DE SOUZA
MARCELO MARQUES GOMES
MARCIO VIEIRA SOARES
RICARDO CONCILIO

Dados Internacionais de Catalogação na Publicação (CIP)
(Câmara Brasileira do Livro, SP, Brasil)

Algoritmos e lógica de programação : um texto introdutório para engenharia / Marco Antonio Furlan de Souza...[et al.]. -- 2. ed rev. e ampl. -- São Paulo : Cengage Learning, 2011.

Outros autores: Marcelo Marques Gomes, Marcio Vieira Soares, Ricardo Concilio
Bibliografia.
ISBN 978-85-221-1129-9

1. Algoritmos 2. Dados - Estruturas (Ciência da computação) I. Souza, Marco Antonio Furlan de. II. Gomes, Marcelo Marques. III. Soares, Marcio Vieira. IV. Concilio, Ricardo.

11-00276 CDD-005.1

Índice para catálogo sistemático:

1. Algoritmos : Computadores : Programação : Processamento de dados 005.1

Algoritmos e Lógica de Programação

Um Texto Introdutório para Engenharia

2ª edição revista e ampliada

Marco Antonio Furlan de Souza
Marcelo Marques Gomes
Marcio Vieira Soares
Ricardo Concilio

Austrália • Brasil • Japão • Coreia • México • Cingapura • Espanha • Reino Unido • Estados Unidos

Algoritmos e Lógica de Programação
Um Texto Introdutório para Engenharia

Marco Antonio Furlan de Souza
Marcelo Marques Gomes
Marcio Vieira Soares e
Ricardo Concilio

Gerente Editorial: Patricia La Rosa

Editor de Desenvolvimento: Fábio Gonçalves

Supervisora de Produção Editorial: Fabiana Alencar Albuquerque

Copidesque: Maria Alice da Costa

Revisão: Rinaldo Milesi

Diagramação: Marco Antonio Furlan de Souza

Capa: Ale Gustavo

Pesquisa Iconográfica: Graciela Naliati

ISBN-13: 978-85-221-1129-9
ISBN-10: 85-221-1129-4

Cengage Learning
Condomínio E-Business Park
Rua Werner Siemens, 111 – Prédio 20 – Espaço 04
Lapa de Baixo – CEP 05069-900 – São Paulo – SP
Tel.: (11) 3665-9900 – Fax: (11) 3665-9901
SAC: 0800 11 19 39

Para suas soluções de curso e aprendizado, visite **www.cengage.com.br**

Impresso no Brasil.
Printed in Brazil.
1 2 3 4 5 6 7   13 12 11 10 09

Para meus pais e para minha esposa Isabel.

M.A.F.S.

Dedico este trabalho aos meus pais, pela dedicação e apoio durante toda a vida.

M.M.G.

À minha esposa Ivani e ao tio Cesar Timo-Iaria pelo apoio e paciência.

M.V.S.

Para todos os amigos que colaboraram, mesmo que indiretamente, com esse trabalho.

Para ela, quem me acompanhará no dia a dia.

R.C.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Prefácio

A quem se destina este livro?

Este livro destina-se a um curso introdutório de lógica de programação, especialmente para aqueles ministrados em escolas de Engenharia. Um dos principais problemas encontrados pelo estudante de Engenharia em um primeiro curso de lógica de programação é a carência de textos que abordem de forma direta e clara as etapas necessárias para suportar o processo de resolução de problemas (computacionais ou não), a saber: a *análise*, com a identificação e solução de subproblemas, e a *síntese*, união das soluções encontradas para compor a solução do problema original. O resultado dessas etapas é sintetizado em passos que devem ser seguidos em determinada ordem e que constituem os *algoritmos*.

Abordagem empregada

Pretende-se aqui seguir uma apresentação incremental dos tópicos. Inicialmente são propostos problemas simples que envolvem raciocínio lógico e que possuem solução livre, de modo a ambientar e a incentivar o estudante na descrição dos passos elementares necessários à resolução de problemas. Isso é fundamental, pois grande parte dos estudantes que tem um primeiro contato com lógica de programação apresenta deficiências na organização de suas soluções e em abstrações. Além disso, neste primeiro contato, um processo genérico de solução de problemas é apresentado de maneira a fornecer um conjunto de dicas ou heurísticas que podem ser aplicadas em todos os problemas a serem resolvidos, fortalecendo assim o processo de abstração, essencial em programação.

A seguir são apresentados os conceitos de computação e computadores. Embora um primeiro curso de lógica de programação possa ser ministrado sem referências a como um computador é organizado e como funciona, verifica-se na prática que esse enfoque não é adequado. Como se sabe, o grande problema do estudante nesses cursos

introdutórios é a abstração de procedimentos e dados. Nesse ponto apresenta-se uma arquitetura de computador bem simples, baseada na arquitetura de Von Neumann, para fixar de modo tangível os conceitos relacionados a instruções e dados operados em computadores. Objetiva-se aqui que o estudante futuramente consiga relacionar os aspectos abstratos de computação, tais como variáveis, estruturas de programas e decomposição funcional com sua implementação. Essa parte serve ainda como incentivo para a necessidade de se descrever algoritmos antes de sua implementação propriamente dita.

Depois, e acompanhando todo o livro, emprega-se uma notação formal para a solução de problemas. Utiliza-se neste texto a descrição de algoritmos sob a forma de *fluxogramas* baseados na norma ISO 5807/1985. Os fluxogramas são compostos por símbolos básicos que representam as menores partes em um processo de solução: estruturas sequenciais, de decisão e de repetição. O uso de fluxogramas nesta obra é justificado pelo fato de que o engenheiro tem a obrigação de desenvolver um raciocínio lógico bem-estruturado e que o fluxograma ainda representa uma poderosa ferramenta para a verificação e teste da lógica empregada na solução de problemas. A utilização de fluxogramas em Engenharia é ampla: de descrições de programas até descrições de processos de fabricação ou processos químicos, seu emprego é similar e regido única e exclusivamente pela lógica utilizada na composição de seus blocos, até se alcançar a solução de um determinado problema.

Além do uso de fluxogramas, são apresentadas ainda duas outras formas conhecidas para a representação de algoritmos: diagramas de *Nassi-Schneidermann* e o pseudocódigo baseado na língua portuguesa, o *Portugol*. Os diagramas de Nassi-Schneidermann empregam uma representação em "caixas" aninhadas, em que cada uma é relacionada a um determinado tipo de comando ou estrutura de programação. Já o Portugol usa uma descrição textual e estruturada da solução de um problema na qual os comandos são descritos por palavras-chave reservadas e extraídas da língua portuguesa.

Descrição dos capítulos

No Capítulo 1 são apresentados os conceitos básicos sobre modelagem de problemas em Engenharia e como organizar suas soluções utilizando passos elementares. Faz-se aqui um prelúdio ao estudo dos algoritmos, com uma descrição de métodos para auxiliar o estudante no processo de identificação e resolução de problemas, bem como a proposição de problemas de lógica com solução livre para ambientar o estudante nesse processo.

No Capítulo 2 são discutidos os conceitos de computação e computadores. Inicia-se com a discussão da origem da palavra computação, seu significado e aplicações. A seguir são discutidos os conceitos básicos sobre a organização de computadores utilizando a arquitetura de Von Neumann. Um computador hipotético com instruções simplificadas é apresentado de forma a proporcionar ao estudante simulações de como as instruções e os dados são realmente processados.

Os conceitos de algoritmo e fluxograma são formalizados no Capítulo 3. São discutidos o conceito e as propriedades de um algoritmo, a representação de algoritmos por fluxogramas, como criar um fluxograma utilizando os símbolos básicos da norma ISO 5807/1985, bem como convenções para os tipos de dados, os nomes de variáveis e operadores.

Já no Capítulo 4 formalizam-se as estruturas de programação. São apresentados os nomes e as topologias das estruturas típicas de um programa: as estruturas sequenciais, de decisão e de repetição. Apresentam-se ainda nesse capítulo duas outras formas de representação de algoritmos: os diagramas de Nassi-Schneidermann e a pseudolinguagem Portugol.

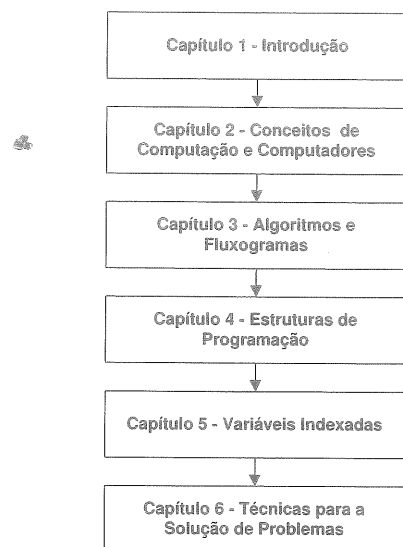
É apresentado, no Capítulo 5, o conceito de variáveis indexadas e seu uso. As variáveis indexadas são aquelas que referenciam de forma ordenada uma sequência de dados homogêneos. Separam-se aqui, para melhor compreensão, o conceito e a utilização de variável indexada unidimensional (ou vetor) do conceito de variável indexada bidimensional e multidimensional. Com essa separação, espera-se que o estudante consiga estender os conceitos e operações relacionados a variáveis indexadas unidimensionais para dimensões maiores.

Por fim, no Capítulo 6 são discutidas as técnicas para a solução de problemas, mais especificamente as técnicas para modularizar a solução utilizando sub-rotinas. São apontados os dois tipos básicos de sub-rotinas (função e procedimento) e como empregá-los de acordo com a técnica *top-down* de modularização.

A Figura da página xxii exibe a organização dos capítulos deste livro.

Convenções tipográficas

Algumas convenções tipográficas foram utilizadas neste livro para tornar mais clara a sua compreensão:

- **Negrito** é empregado para destacar os conceitos importantes.
- *Itálico* é utilizado para enfatizar os conceitos essenciais e para palavras estrangeiras.

- Nos exercícios existem símbolos para identificar aqueles que são básicos, de resolução imediata; médios, nos quais o estudante deve pensar um pouco mais na solução; e desafios, a fim de empenhar-se mais na sua solução:

  ☼ exercício fácil

  ⛅ exercício médio

  ⛈ exercício desafiador

- Exercícios que possuem solução no final do respectivo capítulo são anotados ainda com o símbolo a seguir:

  ✎ exercício com solução

Agradecimentos

Agradecemos a todos os nossos colegas professores envolvidos na disciplina Algoritmos e Programação do Ciclo Básico da Escola de Engenharia Mauá, cujas observações e críticas foram fundamentais: Douglas Lauria, Daniela Caio André, Jorge Kawamura, Lincoln César Zamboni, Paulo Guilherme Seifer, Ricardo Aurélio Roverso Abrão, Vitor Alex Oliveira Alves, Wilson Inacio Pereira e, em especial, Roberto Scalco, pelo auxílio na confecção de algumas figuras deste livro.

Capítulo 1

Introdução

Um programa de computador é um produto resultante da atividade intelectual de um programador. Essa atividade, por sua vez, depende de um treinamento prévio em abstração e modelagem de problemas, bem como o uso da lógica na verificação das soluções. Neste capítulo são apresentados os conceitos introdutórios sobre a tarefa de programar computadores, a necessidade do uso de lógica na programação, a importância de se abstrair e modelar os problemas antes de partir para as soluções. Por fim, são apresentadas algumas dicas úteis que podem ser utilizadas na solução de problemas em geral.

1.1 O desenvolvimento de um software

Um **programa de computador** ou simplesmente **software** é representado pelas **instruções** e **dados** que algum ser humano definiu e que ao serem executados por alguma **máquina** cumprem algum **objetivo**. A máquina a que este texto se refere é um **computador digital**[1].

Os computadores digitais são máquinas eletrônicas contendo processadores e circuitos digitais adequados, operando com sinais elétricos em dois níveis ou **binários** (a ser detalhados no Capítulo 2).

Os dados são organizados em um computador de acordo com sua representação binária, isto é, sequências de **0**s e **1**s. O objetivo de se utilizar um computador é extrair as **informações** resultantes de computações, isto é, o resultado das execuções das instruções de algum programa. Deve-se observar a diferença entre informação e dado: o dado

[1]Existem também computadores analógicos.

por si só é um valor qualquer armazenado em um computador enquanto a informação representa a *interpretação* desse dado, ou seja, qual o seu significado.

Parte dos dados processados durante a execução de um software é fornecida pelo ser humano (ou outra máquina) e denominada dados de **entrada**. Por outro lado, os dados de **saída** são aqueles fornecidos ao ser humano (ou outra máquina) após o processamento dos dados de entrada.

De qualquer forma, é importante notar que o objetivo do software é que motiva sua construção. Este pode ser definido como alguma necessidade humana, por exemplo, um programa para simular o funcionamento de um circuito digital, um programa para comandar um robô em uma linha de montagem, um sistema de gerenciamento de informações em uma empresa, somente para citar algumas. A Figura 1.1 descreve uma simplificação do processo de desenvolvimento de um software.

Figura 1.1 Simplificação do processo de construção de um software.

Nessa figura, o **cliente** especifica exatamente o que o software deve conter. Ele sabe **o que** o software deve conter e realizar, mas regra geral não sabe como. Ele indica o que o software deve contemplar e executar por meio de especificações chamadas **requisitos**.

Entende-se por cliente a entidade que contrata os serviços para a criação de um software, podendo ser uma empresa, pessoa ou ainda uma empresa que, por iniciativa própria, produza e venda seu software livremente (por exemplo, a Microsoft).

No desenvolvimento, os requisitos do cliente são traduzidos em **especificações técnicas** de software pelos **analistas de sistema** ou **engenheiros de software**. O desenvolvimento de um software é tipicamente dividido nas seguintes etapas:

- **Análise**: criam-se especificações que detalham como o software vai funcionar;
- **Projeto**: criam-se especificações que detalham o resultado da análise em termos mais próximos da implementação do software;

- **Implementação**: utilizando-se uma linguagem de programação e as especificações de projeto, o software é construído;
- **Testes**: após a construção do software, são realizados testes para conferir sua conformidade com os requisitos iniciais. O software deve satisfazer a todas especificações do cliente.

Por fim, após os testes o software é implantado na empresa. A implantação pode variar desde uma simples instalação, que dure alguns minutos, até a instalação e testes de integração de diversos softwares, que pode levar semanas. De qualquer forma, o fato de o software estar finalizado e testado não significa que esteja totalmente livre de erros, também denominados *bugs*. Assim, deve-se voltar e tentar identificar a causa dos erros.

Pior que erros de programação, é o caso em que pode acontecer de o software funcionar corretamente, não apresentar erros, mas não realizar o que o cliente esperava. Nesse caso, deve-se retornar à etapa inicial, verificando os requisitos e refazendo todo o ciclo de desenvolvimento novamente.

É um fato que grande parte do investimento feito em um software é gasta na correção de erros do que propriamente na sua elaboração. Daí, surge a necessidade de enxergar o software como o produto de um processo bem-definido e controlado, que atue sobre as suas etapas de desenvolvimento, em outras palavras, um *processo de engenharia de software*.

1.2 Algoritmos e lógica de programação

Como foi brevemente apresentado na Seção 1.1, o software deve ser encarado como um produto de um processo bem-definido e controlado de engenharia. O intuito deste livro não é entrar em detalhes sobre engenharia de software e sim concentrar-se na disseminação de conceitos básicos que viabilizem a especificação correta de softwares, uma etapa imediatamente anterior a sua implementação ou programação.

O estudo de algoritmos e de lógica de programação é essencial no contexto do processo de criação de um software. Ele está diretamente relacionado com a etapa de projeto de um software em que, mesmo sem saber qual será a linguagem de programação a ser utilizada, se especifica completamente o software a ponto de na implementação ser possível traduzir diretamente essas especificações em linhas de código em alguma linguagem de programação como Pascal, C, Java e outras.

Essa tarefa permite verificar, em um nível maior de abstração, se o software está correto ou não. Permite, inclusive, averiguar se o software atenderá às especificações

originalmente propostas. Assim, evita-se partir diretamente para a etapa de implementação, o que poderá ocasionar mais erros no produto final.

1.2.1 O significado de um algoritmo

Um **algoritmo** representa um conjunto de regras para a solução de um problema. Essa é uma definição geral, podendo ser aplicada a qualquer circunstância que exija a descrição da solução. Dessa forma, uma receita de bolo é um exemplo de um algoritmo[2], pois descreve as regras necessárias para a conclusão de seu objetivo: a preparação de um bolo.

Em um bolo, descrevem-se quais serão os ingredientes e as suas quantidades. Depois, quais são as regras para o seu preparo, como a sequência de inclusão dos ingredientes para bater as gemas; cozimento e assim por diante, conforme a Figura 1.2.

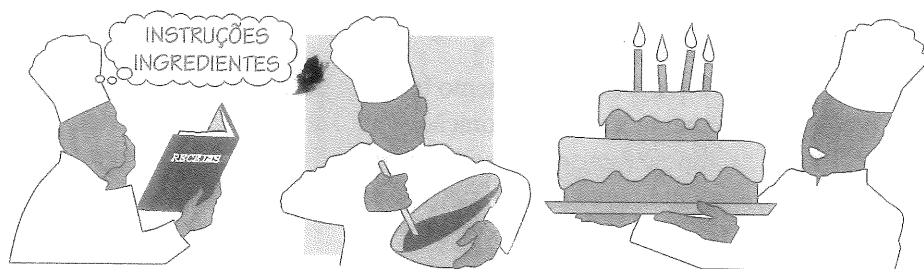



Figura 1.2 Uma receita de bolo é um algoritmo.

A correta execução das instruções contidas na receita de bolo leva à sua preparação. No entanto, se essas instruções tiverem sua ordem trocada ou a quantidade dos ingredientes alterada, o resultado vai divergir do original. Existe, ainda, o perigo de o autor da receita não a ter testado previamente, o que poderá gerar, novamente, resultados indesejáveis[3].

Da mesma forma, em programação, o algoritmo especifica com clareza e de forma correta as instruções que um software deverá conter para que, ao ser executado, forneça resultados esperados (veja a Figura 1.3).

[2]Na realidade, um algoritmo informal e impreciso.

[3]Se o fogão estiver desregulado, também vai gerar problemas.

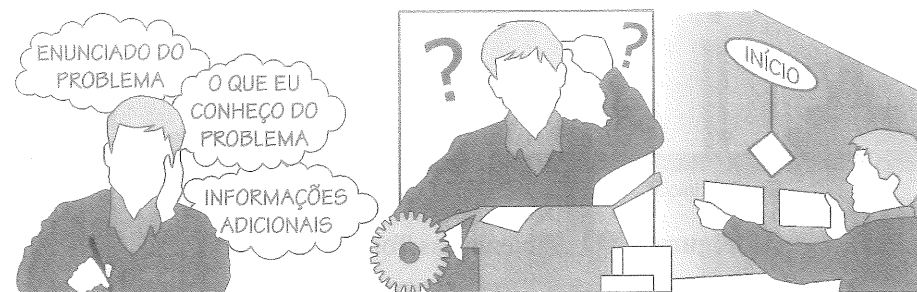



Figura 1.3 A tarefa de especificar um algoritmo.

Em primeiro lugar, deve-se saber qual é o problema a ser resolvido pelo software – o seu objetivo. Daí, deve-se extrair todas as informações a respeito desse problema (dados e operações), relacioná-las com o conhecimento atual que se tem do assunto, buscando eventualmente informações de outras fontes. Essa fase representa a **modelagem** do problema em questão e vai ser detalhada na Seção 1.4.2. A modelagem do problema é resultante de um processo mental de **abstração**, o qual será discutido na Seção 1.4.

Depois, sabendo como resolver o problema, a tarefa consiste em descrever claramente os passos para se chegar à sua solução. Os passos por si só não resolvem o problema; é necessário colocá-los em uma sequência **lógica** (veja Seção 1.4.3), que, ao ser seguida, de fato o solucionará.

Além disso, é importante que essa descrição possua algum tipo de **convenção** para que todas as pessoas envolvidas na definição do algoritmo possam entendê-lo (veja a Seção 1.3). Chega-se, então, na especificação do algoritmo.

1.2.2 Exemplo de algoritmo

Considere o problema das *Torres de Hanoi*. A proposição do problema é a seguinte: inicialmente têm-se três hastes, A, B e C, e na haste A repousam três anéis de diâmetros diferentes, em ordem decrescente por diâmetro (veja a Figura 1.4).

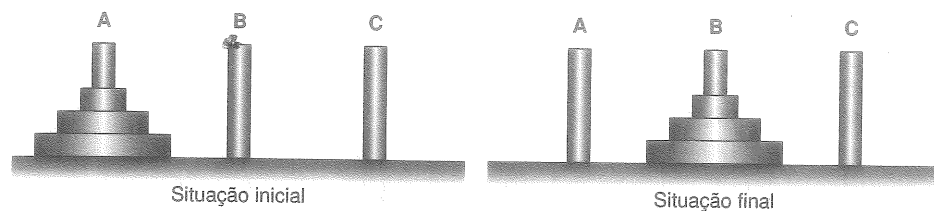



Figura 1.4 O problema das Torres de Hanoi.

O objetivo é transferir os três anéis da haste A para B, usando C se necessário. As regras de movimento são:

- deve-se mover um único anel por vez;
- um anel de diâmetro maior nunca pode repousar sobre algum outro de diâmetro menor.

As únicas informações para se resolver esse problema são as configurações inicial e final dos anéis e as regras de movimento. Uma solução poderia ser a seguinte sequência de operações (veja o Algoritmo 1.1 e a Figura 1.5).

Algoritmo 1.1 Algoritmo para resolver o problema das Torres de Hanoi.

Início

1. Mover um anel da haste A para a haste B.
2. Mover um anel da haste A para a haste C.
3. Mover um anel da haste B para a haste C.
4. Mover um anel da haste A para a haste B.
5. Mover um anel da haste C para a haste A.
6. Mover um anel da haste C para a haste B.
7. Mover um anel da haste A para a haste B.

Fim

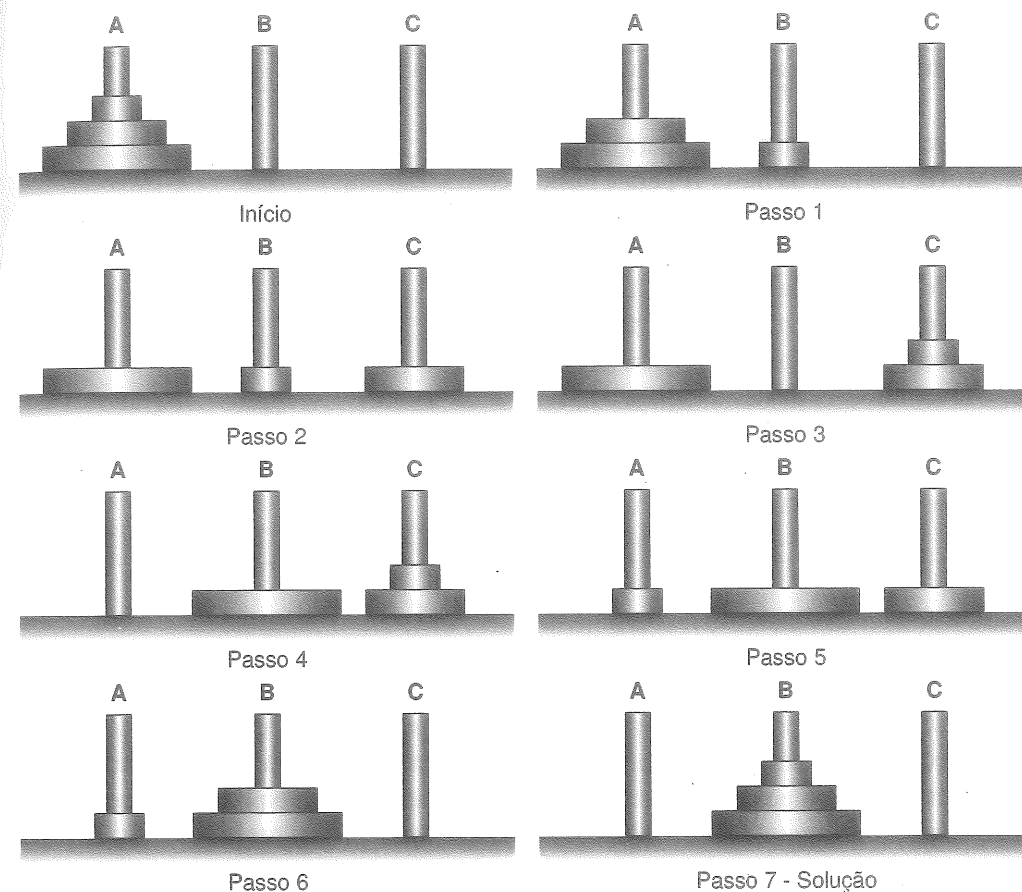



Figura 1.5 Solução do problema das Torres de Hanoi.

Como se obteve essa solução? Primeiro, é importante entender o enunciado do problema e as regras que foram impostas. Dessa forma, não é possível especificar os movimentos nos quais uma peça que esteja abaixo de outra seja movida e nem mover mais de uma peça por vez. Segundo, é importante verificar a cada passo definido se a solução está se aproximando do objetivo final.

O Algoritmo 1.2 define outra sequência de operações para se solucionar o problema.

Algoritmo 1.2 Outro algoritmo para as Torres de Hanoi.

Início

1. Mover um anel da haste A para a haste C.
2. Mover um anel da haste A para a haste B.
3. Mover um anel da haste C para a haste B.
4. Mover um anel da haste A para a haste C.
5. Mover um anel da haste B para a haste C.
6. Mover um anel da haste B para a haste A.
7. Mover um anel da haste C para a haste A.
8. Mover um anel da haste C para a haste B.
9. Mover um anel da haste A para a haste C.
10. Mover um anel da haste A para a haste B.
11. Mover um anel da haste C para a haste B.

Fim

Deve-se observar que essa solução é válida e também resolve o caso das Torres de Hanoi. No entanto, leva-se mais tempo para chegar à solução (onze passos contra sete da solução anterior). Esse exemplo demonstra que uma mesma solução pode ser melhor ou pior que outra. Isso é um conceito importante quando se trata de um programa, pois, dependendo do problema, determinar uma solução mais eficiente pode economizar até horas de processamento. Outras soluções válidas também podem ser propostas, mas não terão menos que sete movimentos.

Agora, e se for considerado o mesmo problema, porém, com n anéis postados inicialmente na haste A? Como isso afetará a solução? É interessante estudar diversos casos particulares antes de elaborar uma solução genérica.

A Tabela 1.1 demonstra alguns resultados do problema das Torres de Hanoi quando se varia n (considere que $X \rightarrow Y$ representa a operação "mover uma peça da haste X para Y").

Tabela 1.1 Alguns resultados para o problema das Torres de Hanoi.

n	**Número de movimentos**	**Movimentos**
1	1	$A \rightarrow B$
2	3	$A \rightarrow C; A \rightarrow B; C \rightarrow B$
3	7	$A \rightarrow B; A \rightarrow C; B \rightarrow C; A \rightarrow B; C \rightarrow A; C \rightarrow B; A \rightarrow B$
4	15	$A \rightarrow C; A \rightarrow B; C \rightarrow B; A \rightarrow C; B \rightarrow A; B \rightarrow C; A \rightarrow C; A \rightarrow B; C \rightarrow B; C \rightarrow A; B \rightarrow A; C \rightarrow B; A \rightarrow C; A \rightarrow B; C \rightarrow B$
5	31	$A \rightarrow B; A \rightarrow C; B \rightarrow C; A \rightarrow B; C \rightarrow A; C \rightarrow B; A \rightarrow B; A \rightarrow C; B \rightarrow C; B \rightarrow A; C \rightarrow A; B \rightarrow C; A \rightarrow B; A \rightarrow C; B \rightarrow C; A \rightarrow B; C \rightarrow A; C \rightarrow B; A \rightarrow B; C \rightarrow A; B \rightarrow C; B \rightarrow A; C \rightarrow A; C \rightarrow B; A \rightarrow B; A \rightarrow C; B \rightarrow C; A \rightarrow B; C \rightarrow A; C \rightarrow B; A \rightarrow B$

É possível notar pela Tabela 1.1 que existe uma relação matemática entre n o número de anéis do problema com o número de movimentos executados, sendo a relação $m(n) = 2^n - 1$, em que m representa os números de movimentos.

Assim, se n for igual a 20, o número de movimentos que deverão ser descritos será igual a 1.048.575 (haja lápis e papel!). Portanto, a generalização do algoritmo desse problema, na forma em que está sendo descrito, é impraticável para grandes valores de n. É necessário determinar formas mais sintéticas para expressar a solução desse estudo de caso.

Um modo de expressar a solução geral desse problema de forma mais sintética é enxergá-lo de outra maneira. Considere que as três hastes estão dispostas em um círculo, conforme ilustrado na Figura 1.6.

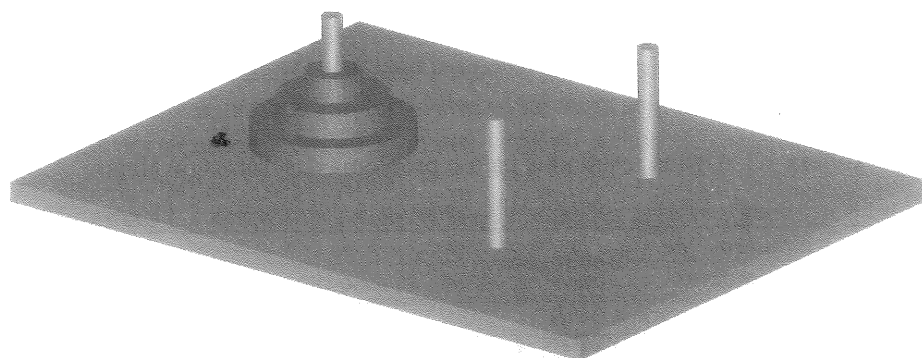



Figura 1.6 Preparação para o uso do algoritmo geral para as Torres de Hanoi.

Um algoritmo que proporciona uma solução geral para o problema das Torres de Hanoi é descrito pelo Algoritmo 1.3. Esse algoritmo não impõe qual deve ser a haste destino. Observa-se que para os valores de n ímpar, os anéis serão transferidos para a primeira haste que estiver após a haste de início, no sentido horário. Se n for par, os anéis serão transferidos para a primeira haste que estiver após aquela de início, no sentido anti-horário.

Algoritmo 1.3 Algoritmo geral para as Torres de Hanoi.

Início

1. **Repita** {*repetir a execução das duas linhas abaixo até que a condição na parte* ***até*** *seja atendida.*}
2. Mova o menor anel de sua haste atual para a próxima, no sentido horário.
3. Execute o único movimento possível com um anel que não seja o menor de todos.
4. **Até** que todos os discos tenham sido transferidos para outra haste.

Fim

A Figura 1.7 mostra a aplicação desse algoritmo para n valendo 3.

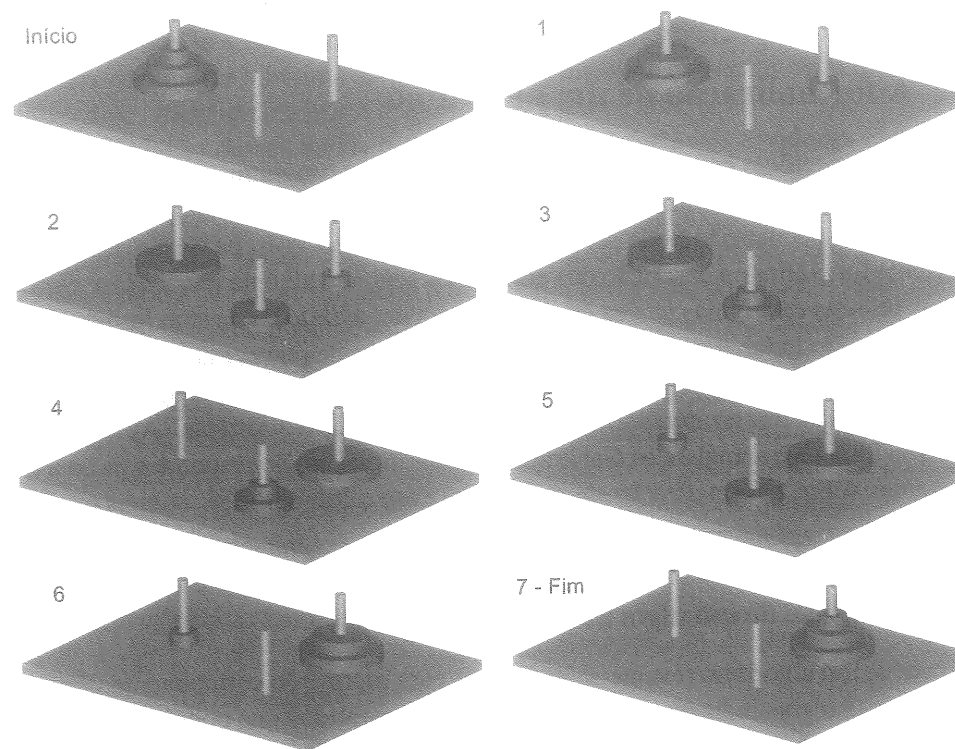



Figura 1.7 Uso do algoritmo geral para o problema das Torres de Hanoi.

Essa solução funciona para qualquer valor de n tal que $n \geq 1$. Apesar de haver um algoritmo geral que resolve todos os casos para o problema das Torres de Hanoi, ainda existem algumas dificuldades na sua forma de descrição:

- Existe um tratamento informal dos comandos: o que significa *repita* e *até*. É necessário formalizar os comandos do algoritmo.
- As operações descritas pelo algoritmo são úteis para uma pessoa interpretar, mas não para uma máquina. É necessário modelar melhor o problema de forma que este seja facilmente traduzido para uma máquina. Está implícito, por exemplo, que não deve ser realizado nenhum movimento no passo 3 se não existir nenhuma outra peça que possa ser movida além do menor anel.

A necessidade de se formalizar um algoritmo é discutida na Seção 1.3 e os tópicos de modelagem de problemas são apresentados na Seção 1.4.2.

1.3 A formalização de um algoritmo

A tarefa de especificar os algoritmos para representar um programa consiste em detalhar os dados que serão processados pelo programa e as instruções que vão operar sobre esses dados. Essa especificação pode ser feita livremente como visto na Seção 1.2.2, mas é importante **formalizar** a descrição dos algoritmos segundo alguma **convenção**, para que todas as pessoas envolvidas na sua criação possam entendê-lo da mesma forma.

Em primeiro lugar, é necessário definir um conjunto de regras que regulem a escrita do algoritmo, isto é, regras de **sintaxe**. Em segundo, é preciso estabelecer as regras que permitam interpretar um algoritmo, que são as regras **semânticas**. Apesar de essa formalização ser assunto para os Capítulos 3 e 4, nesta seção são resumidos os conceitos importantes sobre a formalização de algoritmos com o intuito de já ambientar o leitor nesse tópico.

1.3.1 A sintaxe de um algoritmo

A sintaxe de um algoritmo resume-se nas regras para escrevê-lo corretamente. Em computação, essas regras indicam quais são os tipos de **comandos** que podem ser utilizados e também como neles escrever **expressões**. As expressões de um comando em um algoritmo realizam algum tipo de **operação** com os **dados** envolvidos, isto é, operam com valores e resultam em outros valores que são usados pelo algoritmo.

Os tipos de comandos de um algoritmo são também denominados **estruturas de programação**. Existem apenas três tipos de estruturas que podem ser utilizadas para escrever qualquer programa: **estruturas sequenciais, de decisão** e **de repetição**. Por exemplo, o Algoritmo 1.1 emprega apenas as estruturas sequenciais, pois sua execução é direta, imperativa, não havendo nenhum tipo de condição a ser verificada e nenhum desvio em seu caminho. Já o Algoritmo 1.3 usa uma estrutura de repetição, que possui uma condição que, se for verdadeira, terminará sua execução. Enfatiza-se novamente a vantagem de se ter utilizado essa estrutura, uma vez que ela evita que se tenha de escrever todos os $2^n - 1$ comandos de movimentação dos anéis para um problema da Torre de Hanoi com $n \geq 1$.

As expressões que são escritas em estruturas de programação envolvem a utilização de dados. Antecipando o que será visto no Capítulo 2, os dados em um computador são números binários, isto é, sequências de 0s e 1s, e são armazenados em sua memória.

Não é prático trabalhar diretamente com essa representação e então convencionou-se que os dados manipulados por um programa são categorizados em **tipos de dados**, que torna simples seu uso para o programador, mas que na realidade, para a máquina, são traduzidos em valores binários. Assim é possível manipular diversos tipos de dados em um algoritmo: números inteiros e reais, valores lógicos, textos etc.

A manipulação desses dados é feita por meio de **variáveis** e **valores constantes**, que representam no texto do algoritmo os dados que serão armazenados na memória do computador. O significado de variável é similar àquele empregado na matemática: representar um valor, porém com um significado físico por trás; esse valor será armazenado na memória de um computador. Um valor constante representa um valor que não pode ser alterado, como o número 25, o nome 'Márcio' e assim por diante.

Uma variável pode ser manipulada de muitas formas. Os valores constantes ou resultados de expressões envolvendo as variáveis podem ser **atribuídos** a estas. Para escrever as expressões corretamente, é necessária também uma sintaxe. Essa sintaxe depende do tipo de variáveis envolvidas e determina quais são os **operadores** que podem ser aplicados. Além dos operadores propriamente ditos, especificam-se algumas **funções e procedimentos** predefinidos úteis, que simplificam algumas tarefas corriqueiras em computação, como ler os dados digitados por um usuário, escrever resultados na tela do computador, calcular o seno de um número etc.

1.3.2 Exemplo de sintaxe de um algoritmo

Deseja-se especificar um algoritmo para calcular e exibir na tela a área de um triângulo de base b e altura h, em que os valores de b e de h são fornecidos pelo usuário via teclado. Antes de mais nada, a solução desse problema é imediata, pois sabe-se que a área s de um triângulo de base b e altura h é dada por $s = \frac{b \times h}{2}$.

Um algoritmo para se resolver esse problema pode ser definido de maneira informal, como ilustrado pelo Algoritmo 1.4. Observe que essa descrição está em português e não é conveniente que seja traduzida para uma linguagem de computador.

Algoritmo 1.4 Algoritmo informal para calcular a área de um triângulo.

Início

1. Pedir para o usuário digitar os valores de b e de h.
2. Calcular a área s usando a fórmula $s = \frac{b \times h}{2}$.
3. Exibir o valor de s na tela.

Fim

Agora, a tarefa é descrever esse mesmo algoritmo, utilizando alguma representação mais formal. Primeiro, será considerada a representação por fluxograma (veja Capítulo 3) cujo resultado é apresentado na Figura 1.8.

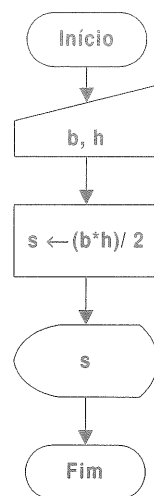



Figura 1.8 Fluxograma para calcular a área de um triângulo.

Nessa representação, a sintaxe para a escrita de um algoritmo é dada pelos **símbolos** do fluxograma e pelas regras para a escrita das expressões. A regra geral de um fluxograma estabelece que este deva ser escrito com seus símbolos básicos, interligados por linhas com ou sem setas que indicam a direção em que os comandos devem ser executados. Sua interpretação deve começar em um símbolo de *início* e terminar em um símbolo de *fim*. O início e fim do algoritmo são representados por dois retângulos de cantos arredondados, que são sempre os mesmos para qualquer fluxograma.

Os dados do problema – a base, a altura e a área – são representados pelas variáveis b, h e s. O símbolo do trapézio representa um comando que possibilita ao usuário digitar os valores que serão atribuídos às variáveis b e h. O símbolo do retângulo representa um comando a ser executado de forma imperativa.

Por sua vez, o comando representado pela flecha esquerda ($\leftarrow$) permite copiar para a variável s o valor da expressão $(b * h)/2$. Observe que existe uma regra para se escrever essa expressão: $*$ significa multiplicação e $/$, divisão. Por fim, o símbolo de um retângulo de cantos arredondados mais achatado à esquerda significa exibir o valor da variável s na tela.

Agora será considerado esse mesmo algoritmo, utilizando-se uma representação em pseudocódigo denominado *Portugol* (veja o Capítulo 4), apresentado pelo Algoritmo 1.5.

Algoritmo 1.5 Algoritmo em Portugol para calcular a área de um triângulo.

Início
1. **Leia**(b, h).
2. $s \leftarrow (b * h)/2$.
3. **Exiba**(s).

Fim

Nesse caso, a sintaxe é dada pelos comandos dessa pseudolinguagem. A regra geral para se escrever um algoritmo nessa representação determina que este deva ser delimitado pelas palavras *início* e *fim* e os comandos, logo após o símbolo de início, devem ser executados sequencialmente, de cima para baixo. O comando *leia* corresponde a um procedimento que automaticamente pede para o usuário digitar dois valores e estes são copiados para as variáveis b e h. A expressão do cálculo de s tem o mesmo significado que no caso do fluxograma, e o comando *exiba* mostra na tela do computador o valor da variável s.

1.3.3 A semântica de um algoritmo

Como já foi exposto, a semântica de um algoritmo estabelece regras para sua interpretação. Os símbolos ou comandos de um algoritmo por si só não têm um significado, a menos que este seja bem-definido.

Por exemplo, no caso de um fluxograma, o símbolo de retângulo representa um comando que deve ser executado de forma imperativa, isto é, sem condição alguma. É definido geralmente como um símbolo e, no seu interior, figura uma expressão que deve ser avaliada, podendo ser qualquer uma desde que seja válida para esse símbolo.

No exemplo do fluxograma da Figura 1.8, a expressão utilizada no interior do retângulo é $s \leftarrow (b * h)/2$, caracterizando-se por ser válida, pois usa os símbolos definidos para um fluxograma, e sua semântica é *multiplicar o valor de b pelo valor de h, dividir esse resultado por 2 e copiá-lo para a variável s.*

Dessa forma, a semântica de um algoritmo sempre acompanha a sua sintaxe, fornecendo um significado. A importância da formalização de um algoritmo, sua sintaxe e semântica podem ser resumidos assim:

- Evitar ambiguidades, pois definem regras sintáticas e semânticas que sempre são interpretadas da mesma forma.
- Impedir a criação de símbolos ou comandos desnecessários na criação de um algoritmo: representam um conjunto mínimo de regras que pode ser utilizado em qualquer algoritmo.
- Permitir uma aproximação com as regras de uma linguagem de programação, fazendo, assim, uma fácil tradução de um algoritmo para sua implementação no computador.

1.4 Como resolver problemas

A criação de um algoritmo é uma tarefa essencialmente intelectual. A partir do enunciado de um problema, deseja-se obter um algoritmo que o resolva. Pode-se afirmar que a tarefa de escrever algoritmos é, portanto, uma tarefa de resolver problemas.

1.4.1 A análise e a síntese de um problema

A resolução de um problema envolve duas grandes fases: a **análise** e a **síntese** da solução.

Na fase de análise, o problema é entendido de forma que se descubra o que deve ser solucionado, quais são os dados necessários e as condições para resolvê-lo se esses dados e essas condições são necessários, insuficientes ou redundantes ou ainda contraditórios e, então, parte-se para a sua **modelagem**, podendo ser enriquecida com o auxílio de equações, desenhos ou gráficos. Como resultado dessa fase, tem-se a elaboração de um **plano de ação**, no qual a experiência em problemas similares vistos anteriormente é utilizada e, também, pode ser necessária a utilização de problemas auxiliares. Nessa fase, faz-se uso direto de processos de **abstração**, o que significa elaborar modelos mentais do problema em questão e do encaminhamento de sua solução.

Na etapa de síntese, executa-se o plano definido na fase de análise, representando os passos por meio de um algoritmo. Aqui, emprega-se uma representação formal, como visto na Seção 1.3. É importante que a solução seja verificada e comprovada corretamente, por meio da execução do algoritmo. Essa execução é feita percorrendo-se o algoritmo do seu início até o seu final, e verificando, a cada passo, se o resultado esperado foi obtido. Caso tenha sido encontrada alguma discrepância, deve-se procurar saber qual foi sua causa e eventualmente analisar novamente o problema, repetindo-se assim esse ciclo até que a solução tenha sido obtida.

1.4.2 Modelagem de problemas

A modelagem (geralmente desprezada) é a principal responsável pela facilidade ou dificuldade da resolução de um problema. Na Matemática e na Engenharia, por exemplo, o uso da **linguagem matemática** é fundamental, principalmente pela eliminação de duplos sentidos que acontecem nessa prática.

O mesmo ocorre na computação, com o emprego de linguagens de descrição de algoritmos (como fluxogramas) e de linguagens de programação (como a linguagem Pascal).

Como um exemplo de modelagem, considere o seguinte problema:

> *Compraram-se 30 canetas iguais, que foram pagas com uma nota de R$ 100,00, obtendo-se R$ 67,00 como troco. Quanto custou cada caneta?*

Este é um problema bem simples, cuja solução pode até ser feita "de cabeça"; porém, como pode ser mostrada a solução? Uma possível resposta:

> *Se eu tinha R$ 100,00 e recebi como troco R$ 67,00, o custo do total de canetas é a diferença entre os R$ 100,00 que eu tinha e os R$ 67,00 do troco. Ora, isto vale R$ 33,00; portanto, esse valor foi o total pago pelas canetas. Para saber quanto custou cada caneta, basta dividir os R$ 33,00 por 30, resultando no preço de cada caneta. Assim, cada caneta custou o equivalente a R$ 1,10.*

Esse raciocínio é matematicamente demonstrado por: seja x o custo de cada caneta, então $quantogastei = 30x$. Como $quantogastei + troco = R\$ 100,00$, tem-se:

$$\begin{aligned} 30x + 67 &= 100 \\ 30x &= 100 - 67 \\ 30x &= 33 \\ x &= \frac{33}{30} \\ x &= 1.1 \quad \square \end{aligned}$$

De uma forma mais curta e universalmente entendida, pode-se também dizer que o caminho pode ser obtido por um algoritmo como o Algoritmo 1.6.

Algoritmo 1.6 Algoritmo inicial para solucionar o problema das canetas.

Início
1. Pegar os valores 30, 100 e 67.
2. Subtrair 67 de 100 e dividir o resultado por 30.
3. Mostrar o resultado final.

Fim

Deve-se observar que esse algoritmo resolve apenas uma **instância particular do problema** das canetas. E para solucionar um **caso geral**, tem-se o seguinte:

> *Compraram-se N canetas iguais, que foram pagas com uma nota de Z reais, obtendo-se Y reais como troco. Quanto custou cada caneta?*

Na solução desse problema mais geral, utiliza-se a experiência que foi adquirida no problema particular e sintetizada pelo Algoritmo 1.6. Nesse caso, basta que sejam fornecidos os valores das variáveis N, Z e Y que a solução do problema é a mesma que a anterior e seu algoritmo pode ser descrito como no Algoritmo 1.7.

Algoritmo 1.7 Algoritmo geral para solucionar o problema das canetas.

Início
1. Ler os valores de N, Y e Z.
2. Subtrair Y de Z e dividir o resultado por N.
3. Mostrar o resultado final.

Fim

No entanto, a proposta apresentada tem uma série de restrições: o que acontecerá se alguém pensar em comprar zero canetas? Ou -3 canetas, faz algum sentido? Para alguém que não possua a forma de interpretar os resultados, isto é possível. Suponha que alguém execute o Algoritmo 1.7 com os seguintes valores: $N = 10$, $Z = 10$ e $Y = 15$. O valor de cada caneta será de $-$R\$ 0.50. Isto faz algum sentido?

Então é necessário que, para a solução geral apresentada pelo Algoritmo 1.7 ser realmente consistente, seja levada em consideração a precondição do problema em que o valor pago pelas canetas seja sempre maior que o troco recebido, que o valor pago e a quantidade de canetas seja sempre maior que zero e que o troco seja maior ou igual a zero.

Em termos matemáticos, podem-se expressar essas condições como $Z > Y$, $N > 0$, $Z > 0$ e $Y \geq 0$. Se essas condições forem todas verdadeiras, então, aplica-se a fórmula conhecida e obtém-se o resultado. Caso contrário, o algoritmo deve terminar sinalizando,

de alguma forma, a razão de seu fracasso. O algoritmo correto para a solução geral das canetas está apresentado pelo Algoritmo 1.8.

Algoritmo 1.8 Algoritmo geral e correto para solucionar o problema das canetas.

Início
1. Ler os valores de N, Y e Z.
2. **Se** $Z > Y$ e $N > 0$ e $Y \geq 0$ e $Z > 0$ **Então**
3. Subtrair Y de Z e dividir o resultado por N.
4. Mostrar o resultado final.
5. **Senão**
6. Exibir a mensagem: "Erro: os valores são inconsistentes!".
7. **Fim Se**

Fim

Essa solução pode ser agora formalizada. Utilizando-se a notação de fluxogramas, ela poderia ser representada como exposta na Figura 1.9. É possível descobrir qual a função de todos os símbolos desse fluxograma ao compará-lo com o Algoritmo 1.8?

1.4.3 O papel da lógica em programação

Lógica é uma área da Matemática cujo objetivo é investigar a veracidade de suas proposições. Considere, por exemplo, o caso em que temos as seguintes proposições:

1. Se estiver chovendo, eu pegarei meu guarda-chuva.
2. Está chovendo.

O que se conclui dessas duas proposições? Parece que a conclusão óbvia é: "eu pegarei meu guarda-chuva".

Essa conclusão seguiu o fato de que existe uma *implicação lógica* na primeira proposição, a qual afirma que "se estiver chovendo" implica "eu pegarei meu guarda-chuva". Essa implicação age como uma "regra", que conduz à dedução do fato. Continuando, e se as proposições fossem:

1. Se estiver chovendo, eu pegarei meu guarda-chuva.
2. Eu peguei meu guarda-chuva.

O que se conclui? A conclusão poderia ser: "é plausível que esteja chovendo"[4].

[4]Não se pode afirmar com precisão que está chovendo só porque você pegou seu guarda-chuva!

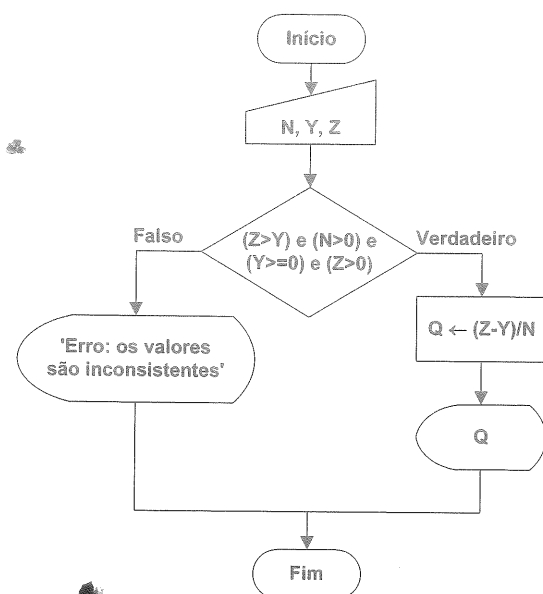



Figura 1.9 Fluxograma para resolver o problema das canetas.

Esses dois exemplos fornecem uma ideia, embora simplificada, do que a lógica se preocupa em estudar. Toda lógica proposta também deve ser formalizada em *elementos sintáticos* (especificam como escrever suas proposições) e *elementos semânticos* (avaliam o significado das proposições – suas interpretações). No caso da lógica clássica, o resultado da avaliação de suas proposições pode ser somente um entre dois valores: **VERDADEIRO** ou **FALSO**. Não se entrará em mais detalhes sobre essa formalização, pois esse assunto está além do escopo deste livro.

O papel da lógica em programação de computadores está relacionado com a correta sequência de instruções que devem ser definidas para que o programa atinja seu objetivo. Serve como instrumento para a verificação do programa escrito, provando se este está correto ou não.

Em um algoritmo em execução, o valor das suas variáveis a cada instante representa o seu **estado**. Com a execução dessas instruções, esse estado vai sendo alterado. Um algoritmo **correto** é aquele que, a partir de um **estado inicial** de suas variáveis, consegue, com a execução de suas instruções, chegar a um **estado final**, no qual os valores das variáveis estão de acordo com a solução esperada.

Voltando-se ao algoritmo geral para a solução do problema das canetas, podemos conferir sua solução analisando a lógica empregada em cada passo. Isso pode ser conferido pelo Algoritmo 1.9, que possui alguns comentários nas instruções.

Algoritmo 1.9 Algoritmo correto, comentado, para solucionar o problema das canetas.

Início
1. Ler os valores de N, Y e Z. {*Nesse ponto temos três valores quaisquer de* N, Y *e* Z.}
2. **Se** $Z > Y$ e $N > 0$ e $Y \geq 0$ e $Z > 0$ **Então** {*Nesse ponto, garante-se que* $Z > Y$, $N > 0$, $Y \geq 0$ *e* $Z > 0$.}
3. Subtrair Y de Z e dividir o resultado por N. {*Como* $Z > Y$, $Y \geq 0$ *e* $Z > 0$ *então* $Z - Y > 0$ *e sendo* $N > 0$*, o resultado existe e é maior que zero.*}
4. Mostrar o resultado final. {*É exibido o resultado, que existe e é maior que zero.*}
5. **Senão**
6. Exibir a mensagem: "Erro: os valores são inconsistentes!". {*Nesse ponto, pelo menos uma das condições do problema não foi atendida.*}
7. **Fim Se**

Fim

No Algoritmo 1.9 provou-se, portanto, que a sua lógica está correta e que leva a resultados esperados. Se forem digitados os valores que atendam às condições do algoritmo, será calculado um valor. Caso contrário, será exibida uma mensagem de erro.

Este é, por conseguinte, o papel da lógica na programação: provar que um algoritmo que foi elaborado está correto. Juntamente com a simulação do algoritmo, que consiste em executá-lo com dados reais, é possível saber se está correto e se leva a valores consistentes.

1.5 Como se portar em um curso de computação

O grande problema apresentado pelos estudantes em um primeiro curso de computação não são as linguagens de programação ou de descrição de algoritmos propriamente ditas, mas sim a dificuldade em abstrair e descrever as soluções de problemas contando apenas com poucas e simples estruturas.

O que deve ser percebido é que o sucesso em um curso ou carreira de computação exige uma predisposição em se envolver com tarefas diretamente intelectuais. Um novo problema de computação pode ser gerado a partir de um já existente, alterando-se apenas poucos elementos de seu enunciado. Isso é facilmente percebido pelos exemplos do problema das Torres de Hanoi (Seção 1.2.2) e pelo problema das canetas (Seção 1.4.2).

Dessa forma, é um **erro decorar** as soluções em computação, pois podem não servir

para outros problemas, que com certeza serão diferentes. O que deve ser procurado é o **entendimento** de como foi obtida uma solução, armazená-la na memória[5] e então utilizar essa experiência adaptando-a a outras situações, por **analogia**, **generalização** ou **especialização**. A grande dica é que um problema pode ser diferente de outro, mas consegue-se aproveitar grande parte da experiência obtida em problemas passados em novos desafios.

Não existe em computação uma "fórmula mágica" para resolver problemas. De qualquer forma, apresenta-se a seguir um conjunto de dicas que podem ser utilizadas durante o processo de raciocínio empregado na resolução de problemas:

1. Ao se deparar com um problema novo, tente entendê-lo. Para auxiliar, pense no seguinte:
    - O que se deve descobrir ou calcular? Qual é o objetivo?
    - Quais são os dados disponíveis? São suficientes?
    - Quais as condições necessárias e suficientes para resolver o problema?
    - Faça um esboço informal de como ligar os dados com as condições.
    - Se possível, modele o problema de forma matemática.
2. Crie um plano com a solução:
    - Consulte sua memória e verifique se você já resolveu algum problema similar. A sua solução pode ser aproveitada por *analogia*, quando o enunciado for diferente, mas a estrutura em si guarda similaridades; por *generalização*, quando se tem uma solução particular e deseja uma solução geral; por *especialização*, quando se conhece alguma solução geral que serve como base para uma em particular ou ainda uma mistura das três técnicas anteriores.
    - Verifique se é necessário introduzir algum elemento novo no problema, como um problema auxiliar.
    - Se o problema for muito complicado, tente quebrá-lo em partes menores e solucionar essas partes.
    - É possível enxergar o problema de outra forma, de modo que seu entendimento se torne mais simples?
3. Formalize a solução:
    - Crie um algoritmo informal com passos que resolvam o problema.

[5] A sua e não a do computador.

    - Verifique se cada passo desse algoritmo está correto.
    - Escreva um algoritmo formalizado por meio de um fluxograma ou outra técnica de representação.
4. Exame dos resultados:
    - Teste o algoritmo com diversos dados de entrada e verifique os resultados (teste de mesa).
    - Se o algoritmo não gerou resultado algum, o problema está na sua sintaxe e nos comandos utilizados. Volte e tente encontrar o erro.
    - Se o algoritmo gerou resultados, estes estão corretos? Analise sua consistência.
    - Se não estão corretos, alguma condição, operação ou ordem das operações está incorreta. Volte e tente encontrar o erro.
5. Otimização da solução:
    - É possível melhorar o algoritmo?
    - É possível reduzir o número de passos ou dados?
    - É possível conseguir uma solução ótima?

Finalizando este capítulo, os problemas de computação são verdadeiros projetos de Engenharia. Aqui se tem a oportunidade de analisar um problema, definir uma estratégia de solução e, por fim, criar um produto, que é o programa em si.

Os programas de computador obtidos não devem ser vistos como objetos isolados do mundo e sim como ferramentas que são empregadas para auxiliar diversas áreas da atividade humana. Dessa forma, este livro utiliza uma abordagem multidisciplinar, na qual os conceitos de outras disciplinas da Engenharia como cálculo, geometria analítica, física etc., são usados nos enunciados dos exercícios, e o objetivo é resolver problemas pertinentes à Engenharia, utilizando o computador como sua ferramenta de trabalho.

1.6 Exercícios

Segue um conjunto de exercícios de lógica que necessitam apenas de raciocínio e bom senso para serem resolvidos. Tente resolvê-los à sua maneira, lembrando-se do que foi discutido neste capítulo. O modo de resolução é livre, mas podem ser utilizadas equações ou frases em português. Faça do seu jeito.

1.1. Descreva como descobrir a moeda falsa em um grupo de cinco moedas, fazendo uso de uma balança analítica (sabe-se que a moeda falsa é mais leve que as outras), com o menor número de pesagens possível. Lembre-se de que sua descrição deve resolver o problema para qualquer situação.

Dica: É possível resolver com apenas duas pesagens.

1.2. Idem ao anterior, porém só se sabe que a moeda falsa tem massa diferente. Para descobrir se ela é mais leve ou mais pesada que as outras, muda-se alguma coisa?

1.3. Idem ao Exercício 1.1, porém com nove moedas.

1.4. Têm-se três garrafas, com formatos diferentes, uma cheia até a boca, com capacidade de oito litros e as outras duas vazias com capacidades de cinco e três litros, respectivamente. Deseja-se separar o conteúdo da primeira garrafa em duas quantidades iguais. Elabore uma rotina que consiga realizar a tarefa, sem que se possa fazer medidas.

1.5. Um caramujo está na parede de um poço a cinco metros de sua borda. Tentando sair do poço, ele sobe três metros durante o dia, porém desce escorregando dois metros durante a noite. Quantos dias levará para o caramujo conseguir sair do poço?

1.6. Um tijolo "pesa" um quilo mais meio tijolo. Quantos quilos "pesam" um tijolo e meio?

1.7. Você está em uma margem de um rio, com três animais: uma galinha, um cachorro e uma raposa. Somente pode atravessar com um animal por vez e nunca deixar a raposa e o cachorro sozinhos nem a raposa e a galinha. Descreva uma forma de conseguir atravessar os três animais, obedecendo a essas condições.

1.8. Você dispõe de uma balança precisa e dez sacos cheios de moedas idênticas na aparência, das quais todas as moedas de um dos sacos são falsas e de massa 1 g menor que as verdadeiras. Qual o menor número de pesagens necessárias para se descobrir o saco de moedas falsas?

1.9. A prova de que $2 = 1$. Considere $a = b = 1$:

$$\begin{aligned} a &= b \\ ab &= b^2 \\ ab - a^2 &= b^2 - a^2 \\ a(b-a) &= (b+a)(b-a) \\ a &= \frac{(b+a)(b-a)}{(b-a)} \\ a &= b + a \\ 1 &= 2 \quad \square \end{aligned}$$

A matemática que você estudou até agora é válida? Então, onde está o erro na dedução anterior?

1.10. Considere o Algoritmo 1.10.

Algoritmo 1.10 Algoritmo para o Exercício 1.10.

Início
1. Ler os valores de A e B
2. $C \leftarrow 0$
3. **Enquanto** $A > B$ **Faça**
4. Subtraia B de A, coloque o resultado em A e some 1 em C
5. **Fim Enquanto**
6. Mostre os valores finais de C e de A

Fim

Execute essas instruções para os seguintes pares de números: 10 e 2, 6 e 2, 15 e 3. O que significa o valor final de C? E o valor final de A?

1.11. Dois amigos se encontraram em uma rua. Eles não se viam há alguns anos. Um dos amigos, aproveitando que o outro é um professor de Matemática, inicia o seguinte diálogo:

– "Já que você é um professor de Matemática, vou lhe dar uma charada. Hoje meus três filhos celebram seus aniversários e eu gostaria que você adivinhasse suas idades".
– "Ok", respondeu o professor. "Mas você precisa me dizer algo sobre eles!".
– "Bem, a primeira dica é que o produto de suas idades é 36".

– "Hum. Só isso não dá para resolver. Preciso de mais alguma dica".
– "A outra dica é que a soma de suas idades é igual ao número de janelas daquele edifício", responde apontando para um edifício próximo.
– O matemático então responde: "Ainda necessito de mais uma ajuda".
– "Bem, meu filho mais velho tem olhos azuis".
– "Já sei quais são as idades", respondeu o matemático.

Seguindo o mesmo raciocínio do matemático deste exercício, descubra quais são as três idades dos filhos de seu amigo.

1.12. Determine quais são os possíveis números (no intervalo fechado de 0 a 9) que se substituídos nos símbolos F, I, A, T torna a multiplicação a seguir verdadeira:

$$\begin{array}{r} IF \\ \times\ AT \\ \hline FIAT \end{array}$$

Os valores de F, I, A, T devem ser diferentes entre si.

Capítulo 2

Conceitos de Computação e Computadores

O estudo de algoritmos e de lógica de programação é facilitado com o conhecimento do funcionamento do computador e de como ele executa seus programas. O objetivo deste capítulo é discutir os conceitos básicos de computação e computadores, introduzindo os conceitos importantes, tais como a organização básica de um computador, como os dados e as instruções são armazenados e como as instruções são executadas. A aplicação desses conceitos será feita a partir do Capítulo 3 e se estenderá por todo o livro.

2.1 Origens da computação

2.1.1 A necessidade de calcular

A capacidade do ser humano em realizar cálculos surgiu com sua habilidade de se comunicar com mais precisão. Inicialmente os primeiros humanoides comunicavam-se por um conjunto de grunhidos, tal como observado em outros animais. A evolução permitiu que houvesse um aprimoramento de suas capacidades cognitivas, levando dessa forma ao surgimento de vocabulários mais extensos e depois à elaboração de regras para a composição de frases com esses vocabulários. Assim, surgiram as primeiras linguagens.

Acompanhando esse processo evolutivo, houve o aparecimento da escrita. A escrita permitiu o registro de informações importantes, que poderiam ser compartilhadas com outros humanos, mesmo que seu autor estivesse ausente. De início, eram pinturas rupestres e marcas em cavernas e em ossos. Depois, apareceram os símbolos que representavam palavras utilizadas na comunicação. Por fim, houve o surgimento do alfabeto,

permitindo que a capacidade de comunicação fosse estendida e simplificada, pois agora não seria mais necessário decorar um número muito grande de símbolos, já que cada palavra pode ser formada a partir de um pequeno alfabeto.

É nesse contexto evolutivo da comunicação humana que surge a necessidade do homem em realizar cálculos. Como o desenvolvimento da capacidade de comunicação é resultado direto do processo de desenvolvimento de raciocínio, o ser humano passou a ter necessidade de controlar e proteger suas atividades primárias. Essas atividades traduziam-se principalmente no registro de suas transações comerciais, como a contagem de rebanhos, a troca de moedas e a divisão de terras, bem como a elaboração de calendários para determinar as estações do ano para a agricultura.

2.1.2 O desenvolvimento de sistemas de numeração

Para representar as quantidades envolvidas em computações, foi necessário o desenvolvimento de sistemas de numeração. No caso do ser humano, o sistema mais evidente é o decimal, em razão do uso de todos os seus dedos para computar (ainda aprendemos assim!). Dessa forma, surgiu a palavra *digitus*, que, traduzindo do latim, significa **dedo**. Em português essa palavra é conhecida como **dígito**.

O sistema decimal emprega a base 10, isto é, cada dedo representa um número no intervalo de 1 a 10. A grande deficiência do sistema decimal empregando as mãos é a capacidade de representar grandes números. Algumas pessoas tentaram explorar esse limite, como o monge beneditino Beda (673-735), que desenvolveu um método que permitia a descrição de números até 10.000, de acordo com a posição dos dedos e um método que possibilitava a contagem até 1.000.000, colocando a mão em várias partes do corpo.

Deve-se observar que nem sempre a base 10 foi utilizada por todos os povos. Os babilônios (2000 a.C.), por exemplo, empregavam o sistema sexagesimal (base 60), pois era baseado em um sistema de unidades de pesos e medidas adotado. Os maias ($\sim$ 0 a.C.) utilizavam um sistema vigesimal (base 20), que derivava da forma como calculavam seus calendários. Os gregos usavam um sistema misto, decimal e hexadecimal. Os romanos, um sistema decimal com símbolos especiais para os números 5, 50 e 500 etc.

Como se pode observar, a contagem utilizando apenas as mãos não é prática. Além do problema de representar grandes números, não é possível registrar os cálculos. Foi então necessário o desenvolvimento de símbolos para representar números escritos. Por exemplo, os egípcios (3500 a.C.) utilizavam um sistema de representação numérica com símbolos específicos para as potências de 10, como 1, 10, 100, 1.000, 10.000 etc. Nesse sistema, o número 1 era representado por um traço vertical; o número 10, por um osso de calcanhar invertido; o número 100, por um laço; o número 1.000, por uma flor de lótus; e o número 10.000, por um dedo dobrado. Esses símbolos estão indicados na Tabela 2.1.

Tabela 2.1 Alguns símbolos do sistema de numeração egípcio.

Número	Símbolo
1	
10	
100	
1.000	
10.000	

Assim, um número como 23.523 primeiro deve ser decomposto em potências de 10 como $2 \times 10^4 + 3 \times 10^3 + 5 \times 10^2 + 2 \times 10^1 + 3 \times 10^0$. Daí, basta repetir o símbolo correspondente a cada potência de acordo com seu fator multiplicativo, conforme apontado na Figura 2.1.

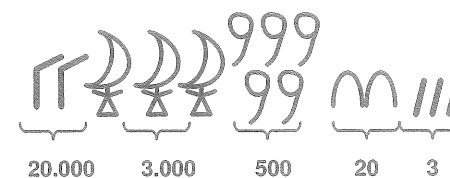



Figura 2.1 O número 23.523 em egípcio.

O sistema empregado pelos romanos era similar. Como já foi citado, eles usavam os símbolos para as potências de 10 e para os números 5, 50, 500 etc. A Tabela 2.2 exibe os símbolos de números utilizados pelos romanos.

Os números começando com 4 e 9 eram formados utilizando-se uma abreviação subtrativa, isto é, 9 igual a 10 menos 1, que em romano era escrito como IX e 40 igual a 50 menos 10, escrito como XL. Seguindo esse raciocínio, um número como 1.979 era escrito assim: $MCMLXXIX$.

Esse tipo de sistema de numeração, embora superior ao uso dos dedos para fazer a contagem, apresenta ainda alguns problemas, como realizar multiplicações. Os romanos, por exemplo, utilizavam-se de um *ábaco* para efetuar as operações aritméticas.

Tabela 2.2 Símbolos do sistema de numeração romano.

Número	Símbolo
1	*I*
5	*V*
10	*X*
50	*L*
100	*C*
500	*D*
1.000	*M*
10.000	$\overline{X}$
100.000	$\overline{C}$
1.000.000	$\overline{M}$

Os primeiros ábacos que se tem notícia datam de aproximadamente 1.000 anos antes de Cristo, sendo utilizados por babilônios e egípcios. Um ábaco é uma placa contendo sulcos (ábaco romano) ou uma esquadria de madeira contendo arames (ábaco japonês ou *soroban*) em que pedrinhas podiam ser movidas, representando dígitos de um número e que permitiam a realização de operações aritméticas (veja a Figura 2.2).

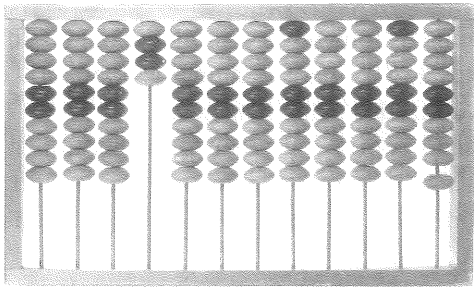



Figura 2.2 Um ábaco típico.

Foi com o uso de pedrinhas para auxiliar nas contagens que surgiu o termo cálculo. O **cálculo**, palavra que dá origem ao verbo calcular, deriva da palavra latina *calculus* e está relacionada com a palavra grega *chalix*, ambas significando pedrinha ou seixo, que foram os primeiros objetos a auxiliar o homem em seus cálculos. Por sua vez, a **computação** significa o ato ou o efeito de **computar**, que é um verbo que exprime o ato de fazer contagem, de contar ou calcular.

O uso do ábaco é bem simples. Cada pedrinha na parte superior vale 5 e na parte inferior vale 1. As colunas (sulcos ou arames) indicam da direita para a esquerda as unidades, dezenas, centenas, milhares etc. Um dígito em sua posição é representado pelo número de pedrinhas que são deslocadas (para cima ou para baixo, de acordo com a convenção adotada) da sua posição original. Por exemplo, a Figura 2.2 representa o número 23.516.

É possível realizar todas as operações aritméticas com um ábaco (algumas podem requerer "truques" particulares). A adição, a operação mais simples de se executar com um ábaco, por exemplo, é feita da seguinte forma: primeiro, representa-se no ábaco o primeiro operando, da forma descrita anteriormente. Depois, sem apagar o primeiro operando, representa-se da mesma forma o segundo operando. Quando ocorre o estouro de uma coluna, é colocado o "vai-um" necessário na próxima. Interpretando o estado final do ábaco, tem-se o resultado.

O problema da utilização do ábaco é que cada passo apaga o precedente, de modo que para se verificar um resultado é necessário refazer o cálculo. No entanto, esse instrumento ainda é muito popular no Japão, onde, em 1946, foi utilizado para derrotar uma calculadora elétrica em uma competição.

Um sistema de numeração próximo ao que se utiliza atualmente foi inventado pelos chineses. Esse sistema emprega a posição dos números para indicar seu valor. Os chineses contavam os números de 1 a 9, representando-os por palitos convenientemente arranjados, conforme exibido na Tabela 2.3.

Os chineses realizavam as operações aritméticas com esses símbolos, colocando os palitos que representam os dígitos em um tabuleiro denominado *suan-phan*. Nesse tabuleiro, uma casa vazia representava 0 (ainda não havia o conceito de 0) e os palitos significando os dígitos eram colocados nas casas de acordo com a posição desse dígito no número. Por exemplo, a soma de 62.014 com 74.158 em um *suan-phan* é representada pela Figura 2.3.

Nessa figura, os números 62.014 e 74.168 estão representados respectivamente na terceira e na quarta linhas. Na quinta linha, tem-se o resultado (136.182) e, na segunda, aparecem os "vai-um" da soma.

Tabela 2.3 Símbolos do sistema de numeração chinês.

<table>
<tr><th>Número</th><th>Símbolo</th></tr>
<tr><td>1</td><td>𝍠</td></tr>
<tr><td>2</td><td>𝍡</td></tr>
<tr><td>3</td><td>𝍢</td></tr>
<tr><td>4</td><td>𝍣</td></tr>
<tr><td>5</td><td>𝍤</td></tr>
<tr><td>6</td><td>𝍥</td></tr>
<tr><td>7</td><td>𝍦</td></tr>
<tr><td>8</td><td>𝍧</td></tr>
<tr><td>9</td><td>𝍨</td></tr>
</table>

<table>
<tr><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>𝍠</td><td></td><td></td><td></td><td>𝍠</td><td></td></tr>
<tr><td></td><td>𝍥</td><td>𝍡</td><td></td><td>𝍠</td><td>𝍣</td></tr>
<tr><td></td><td>𝍦</td><td>𝍣</td><td>𝍠</td><td>𝍥</td><td>𝍧</td></tr>
<tr><td>𝍠</td><td>𝍢</td><td>𝍥</td><td>𝍠</td><td>𝍧</td><td>𝍡</td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td></tr>
</table>

Figura 2.3 O uso do *suan-phan* chinês.

Por fim, é importante mencionar a contribuição da Índia no desenvolvimento dos sistemas de numeração. Foram os hindus que primeiro registraram os cálculos em papéis (650 d.C.) e que inventaram o símbolo para o zero em sua escrita. O uso do zero permitiu que os cálculos pudessem ser representados no papel, sem a necessidade de deixar um espaço em branco.

A matemática hindu foi levada pelos árabes, que a disseminaram pelo Ocidente até a Espanha. Os árabes foram os responsáveis também pela representação numérica que se utiliza atualmente, com a criação do *algarismo*, da *tabuada* e da *álgebra*. Algarismo é uma palavra derivada do nome do matemático *Al-Khwarizmi*[1]. Com as Cruzadas, foi possível introduzir esses conhecimentos no mundo ocidental, os quais são utilizados até hoje.

2.2 A evolução dos computadores

2.2.1 Geração zero – Computadores puramente mecânicos

Com o Iluminismo (século XVIII), principalmente após o surgimento do cálculo diferencial por Newton e Leibniz e da tentativa de definir uma linguagem matemática universal por Leibniz, que mais tarde convergiria para o que se conhece por *lógica simbólica*, surgiram os primeiros dispositivos mecânicos de cálculo.

A primeira calculadora portátil foi desenvolvida por John Napier em 1612 e chamava-se *ossos de Napier* (veja a Figura 2.4).

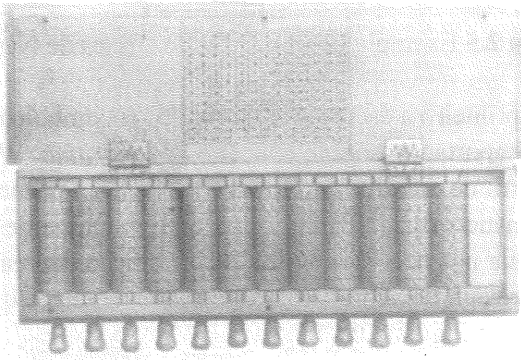



Figura 2.4 Os ossos de Napier.

[1] A palavra algoritmo também deriva de seu nome.

Os ossos de Napier eram na verdade um conjunto de bastões para se realizar as multiplicações por meio de adições. Cada bastão continha a tabuada de um número e a multiplicação de um número x por um número y era realizada consultando-se a x-ésima linha do bastão correspondente ao número y. Por exemplo, para multiplicar o número 3 por 849 bastava procurar na quinta linha dos bastões correspondentes aos números 8, 4 e 9 o valor do resultado das multiplicações, deslocar à esquerda os valores encontrados de acordo com a sua posição no número (sua potência de dez) e então somar os resultados, conforme indicado na Figura 2.5. Assim, o valor de 3×849 é dado pela soma $2.400 + 120 + 27$, ou seja, 2.547.

Índice	8	4	9
1	0/8	0/4	0/9
2	1/6	0/8	1/8
3	2/4	1/2	2/7
4	3/2	1/6	3/6
5	4/0	2/0	4/5
6	4/8	2/4	5/4
7	5/6	2/8	6/3
8	6/4	3/2	7/2
9	7/2	3/6	8/1

Figura 2.5 Exemplo de utilização dos ossos de Napier.

Napier ainda foi o primeiro a escrever números com o símbolo de ponto como separador decimal e, mais importante, criou o conceito de *logaritmo*. William Oughtred, em 1622, deu origem, a partir de conceitos dos ossos de Napier, à primeira régua de cálculo.

Em 1642, o matemático Blaise Pascal criou uma máquina de somar, a *Pascaline*, inicialmente para auxiliar nos negócios de seu pai. A adição era realizada com o auxílio de engrenagens que giravam de acordo com um seletor de disco que permitia escolher um dígito desejado (veja a Figura 2.6).

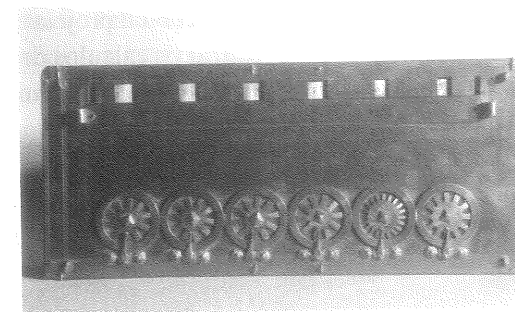



Figura 2.6 A *Pascaline* de Pascal.

O resultado era apresentado em um visor mecânico, acima dos seletores. Em 1673, utilizando uma engrenagem cilíndrica de passo, Leibniz inventou uma máquina que conseguia fazer multiplicações, por somas sucessivas que eram automaticamente realizadas pelas engrenagens.

Outro fator histórico para contribuir no desenvolvimento de dispositivos automáticos de cálculo foi a Revolução Industrial. Nesse contexto, destaca-se inicialmente Joseph-Marie Jacquard, que em 1801, na França, inventou uma máquina de tear automática (veja a Figura 2.7), cujos padrões eram fornecidos por cartões perfurados.

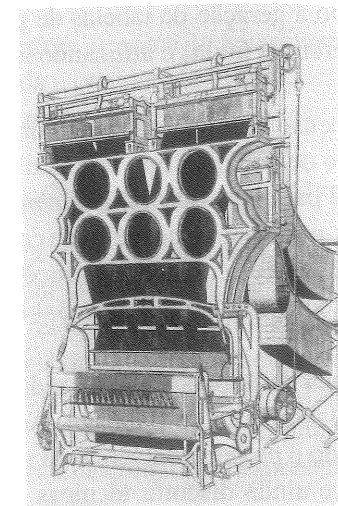



Figura 2.7 O tear automático de Jacquard.

Charles Babbage, em 1822, na Inglaterra, começou a projetar uma máquina a vapor programável, a *máquina de diferenças*, para realizar os cálculos de tabelas de navegação, que apresentavam sérias discrepâncias nas operações (veja a Figura 2.8).

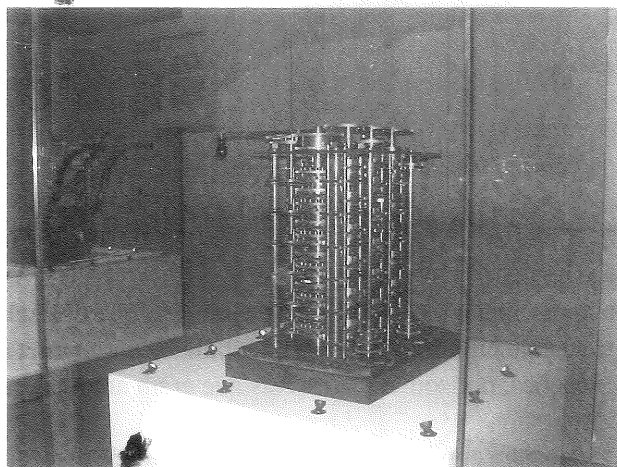



Figura 2.8 A máquina de diferenças de Babbage.

A máquina de diferenças era capaz de realizar somente as adições. No entanto, os cálculos mais sofisticados, como a geração de tabelas de polinômios, podiam ser feitos com o auxílio da técnica de diferenças finitas e utilizando um conjunto finito de posições de memória dessa máquina.

Dez anos depois, Babbage pensou em generalizar o conceito de sua máquina de diferenças para suportar a realização de qualquer tipo de cálculo, mesmo que ela não tivesse sido construída especificamente para isso. Denominada *máquina analítica*, tinha como princípio básico a programação: utilizando o conceito de cartões perfurados de Jacquard, a máquina seria alimentada com cartões contendo instruções e dados que seriam, então, processados pela máquina. Foi com esse projeto que Babbage ficou conhecido como o pai da computação.

Apesar de a máquina analítica de Babbage não ter sido concluída por descrédito de seus financiadores, a grande colaboradora de Babbage, Ada Augusta King, a condessa de Lovelace, além de ter traduzido o projeto conceitual da máquina para a língua inglesa, propôs programas de exemplo e ainda discutiu técnicas de programação para aquela máquina. Ada Augusta tornou-se, então, a primeira programadora do mundo.

Outra contribuição contemporânea, mas de caráter matemático, foi o desenvolvimento de um sistema de lógica simbólica e de raciocínio feito por George Boole, em

1854. A *lógica booleana*, como ficou conhecida, é até hoje usada para o projeto de circuitos digitais utilizados nos computadores.

Chegando próximo ao século XX, em 1890, o norte-americano Hermann Hollerith projetou um equipamento para auxiliar na realização do censo daquele ano. A máquina, chamada de *tabulador eletromecânico*, processava automaticamente cartões perfurados, permitindo assim a contagem do número de habitantes (veja a Figura 2.9).

Figura 2.9 O tabulador eletromecânico de Hollerith.

A partir dessa máquina, surge o nome *processamento de dados*. Com o sucesso de sua máquina, Hollerith funda a companhia CTR (Computing-Tabulating-Recording), que, em 1924, passa a se chamar *International Business Machine* ou apenas IBM.

2.2.2 Primeira geração – Computadores a válvula e relé

Os primeiros computadores eletrônicos começam a surgir na década de 1930. Entre 1935 e 1938, Konrad Zuse, em Berlim, projetou e construiu uma série de máquinas eletromecânicas baseadas em relés. Um relé é um dispositivo que, se excitado por uma corrente elétrica, é capaz de fechar um contato, servindo assim como uma chave "liga-desliga". As máquinas reconhecidas como *Z-1*, *Z-2*, *Z-3* e *Z-4* (veja a Figura 2.10) só foram conhecidas fora da Alemanha após o término da Segunda Guerra Mundial. Essas máquinas utilizavam aritmética binária e já apresentavam uma organização interna similar à dos computadores modernos.

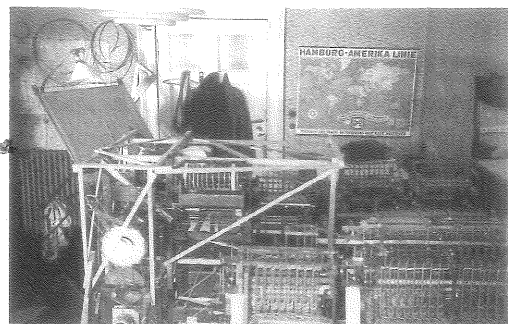



Figura 2.10 O computador Z-1 de Zuse.

Paralelamente, nos Estados Unidos, entre 1936 e 1939, John Vincent Atanasoff e John Berry desenvolveram uma máquina baseada em válvulas, denominada *ABC*, com o propósito de resolver conjuntos de equações lineares da Física (Figura 2.11).

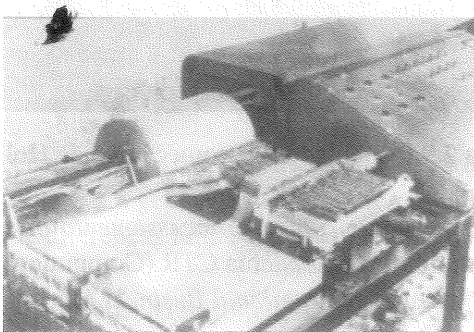



Figura 2.11 O computador ABC de Atanasoff e Berry.

A válvula é um dispositivo puramente eletrônico que, como o relé, funciona como uma chave, porém com velocidade dez mil vezes mais rápida. Da mesma forma que as máquinas de Zuse, a máquina ABC apresentava alguns conceitos encontrados em computadores modernos, como uma unidade aritmética e uma unidade de memória regenerativa, e operava com aritmética binária.

Nessa época, uma grande contribuição teórica foi dada pelo matemático inglês Alan Turing. Ele definiu o conceito intitulado *Máquina Universal de Turing*, estabelecendo um dispositivo teórico capaz de executar qualquer algoritmo descrito, definindo assim as bases para o estudo da computabilidade: um algoritmo computável é aquele que pode ser executado por uma máquina de Turing.

No período da Segunda Guerra Mundial, o computador torna-se uma ferramenta necessária para auxiliar no cálculo de tabelas de balística para canhões navais e artilharia antiaérea. Nos Estados Unidos, destaca-se nesse período o computador eletromecânico Harvard Mark-1 de 1944 (veja a Figura 2.12), concebido por Howard Aiken e implementado pela IBM como ASCC (Automatic Sequence Control Calculator) .

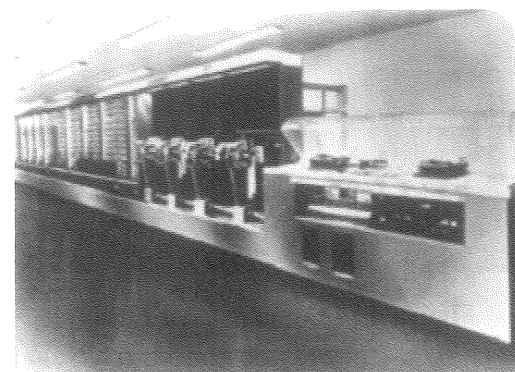



Figura 2.12 O computador Harvard Mark-1 de Aiken.

Essa máquina não possuía o conceito de programa armazenado: o programa era "carregado" por meio de uma fita perfurada, executando as instruções durante sua leitura. Ocupava 120 m^2, continha milhares de relés e conseguia multiplicar números de dez dígitos em três segundos.

Na Inglaterra, outro grande problema era a decifração de códigos secretos alemães. Um projeto secreto de computador denominado *Colossus* (veja a Figura 2.13), foi desenvolvido entre 1940 e 1944 com o intuito de auxiliar na quebra de códigos da máquina alemã Enigma e foi revelado somente em 1970.

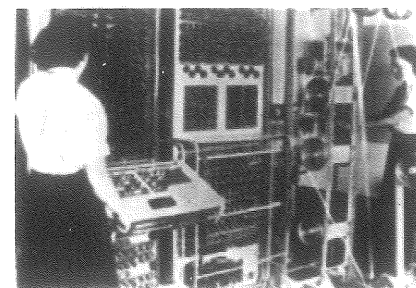



Figura 2.13 O computador britânico Colossus.

Em 1946 é apresentado o computador Eniac (Electronic Numeric Integrator and Calculator), cujo desenvolvimento iniciou-se em 1943, liderado por J. Presper Eckert e John Mauchly (veja a Figura 2.14).

Figura 2.14 O computador Eniac.

O Eniac continha 18 mil válvulas, pesava 30 toneladas e era capaz de realizar 5 mil adições e subtrações e 300 multiplicações por segundo. Possuía uma memória pequena e seus programas eram configurados por cabos, o que tornava complexa a tarefa de programar essa máquina.

Em 1945, nos Estados Unidos, o matemático húngaro John von Neumann, consultor do projeto Eniac, propôs uma arquitetura que seria seguida por todas as gerações de computadores (mais detalhes na Seção 2.4). Ele propôs o conceito de *programa armazenado*, ou seja, a memória do computador armazenaria tanto as instruções a ser executadas quanto os dados a ser processados. Dessa forma, as instruções poderiam ser facilmente modificadas sem a necessidade de alterar as ligações com os cabos ou outros dispositivos. Outro benefício desse conceito é que tanto as instruções quanto os dados seriam armazenados segundo uma única representação, de modo que as instruções seriam executadas da mesma forma que os dados, permitindo, assim, modificações automáticas dessas instruções.

Essa arquitetura dividia o computador em *unidade central de processamento*, *memória principal* e *dispositivos de entrada e saída*, e ficou conhecida como a *arquitetura de Von Neumann*, sendo utilizada inicialmente no computador Edvac (Electronic Discrete Variable Computer), que se tornou operacional em 1951 (veja a Figura 2.15), no computador IAS (Institute for Advanced Study, da Universidade de Princeton), de 1952 (veja a Figura 2.16) e no computador Edsac (Electronic Delay Storage Automatic Calculator) da Universidade de Cambridge, em 1949 (veja a Figura 2.17).

Figura 2.15 O computador Edvac.

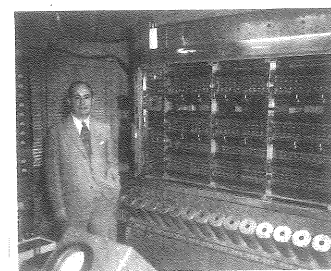



Figura 2.16 John von Neumann e o computador IAS.

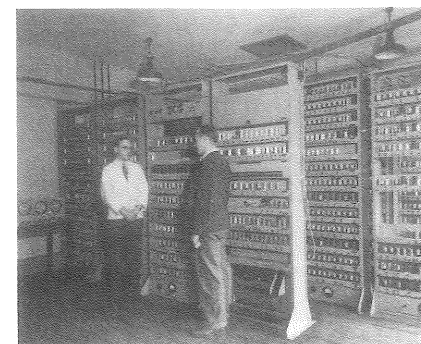



Figura 2.17 O computador Edsac.

Esses desenvolvimentos desencadearam, no início dos anos 50, a criação de outros computadores baseados na arquitetura de Von Neumann. Destes, o primeiro computador que obteve sucesso comercial nessa época foi o Univac (Universal Automatic Computer), de 1951 (veja a Figura 2.18).

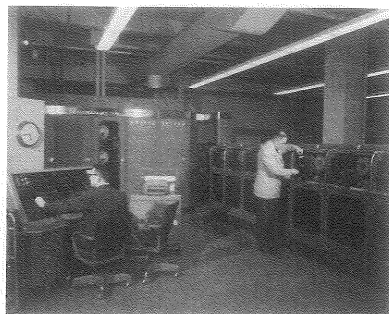



Figura 2.18 O computador Univac.

Em vez de válvulas, o Univac empregava diodos de cristal, que tornava sua velocidade superior aos contemporâneos valvulados. Além disso, foi o primeiro computador a contar com unidades de equipamentos periféricos independentes, como teletipos e impressoras, além de uma sofisticada unidade de armazenamento em fita.

A IBM demorou um pouco mais para lançar outros computadores após o ASCC. Em 1953, a empresa lançou o modelo 701, para processamento científico. Em 1956, o modelo 704, que tinha o dobro de memória do 701 e contava com hardware para realização de cálculos com ponto flutuante, e, em 1958, seu último computador valvulado, o modelo 709, similar ao 704 (veja a Figura 2.19).

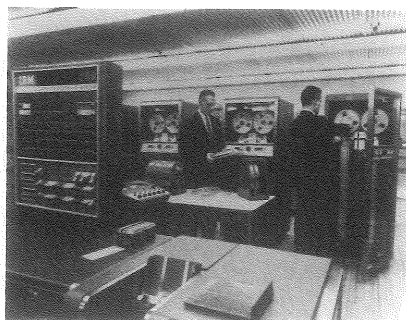



Figura 2.19 O computador IBM 709.

2.2.3 Segunda geração – Computadores transistorizados

Com a invenção do *transistor* em 1947 por William Shockley, John Bardeen e Walter Brattain, foi possível revolucionar a construção de computadores, oferecendo mais confiabilidade e velocidade do que as válvulas. Um transistor é um dispositivo semicondutor, isto é, conduz corrente elétrica de acordo com uma tensão aplicada e por isso pode ser utilizado como uma chave, como a válvula e o relé. Em comparação com a válvula e o relé, o transistor é mais confiável, menor e mais rápido. No entanto, essa tecnologia demoraria dez anos até figurar em um computador.

O primeiro computador transistorizado da história foi o *TX-0*, construído no Massachusetts Institute of Technology, em 1957, para servir como base de testes para o computador *TX-2*. Apesar de o *TX-2* não ter obtido êxito, um engenheiro do MIT, chamado Ken Olsen, fundou uma companhia para manufaturar uma versão comercial do *TX-0*, a DEC (Digital Equipment Company), que, em 1961, lançou o PDP-1, o primeiro minicomputador comercial. O sucesso do PDP-1 fez com que a DEC lançasse o PDP-8 em 1965 e depois outras famílias baseadas nesse modelo até 1990.

A IBM, por sua vez, lançou os modelo 7090 e 7094, que eram as versões transistorizadas baseadas no modelo 709. Apesar de serem muito mais rápidos que seu contemporâneo da DEC, o PDP-1, eram muito mais caros. A IBM então criou o modelo 1401, mais barato e tão rápido quanto os modelos 7090 e 7094, sendo destinado principalmente a aplicações comerciais (veja a Figura 2.20).

Figura 2.20 O computador IBM 1401.

O ápice dos computadores transistorizados foi o CDC-6600, um supercomputador criado pela CDC (Control Data Corporation) em 1964 (veja a Figura 2.21).

Figura 2.21 O computador CDC-6600.

O CDC-6600 distinguia-se dos demais de sua época por descarregar o processamento da CPU pelo uso de pequenos computadores auxiliares que tratavam de outras tarefas como entrada e saída de dados e gerenciamento de tarefas de forma paralela. Dessa forma, conseguia executar até dez instruções simultaneamente.

2.2.4 Terceira geração – Computadores com circuitos integrados

A evolução natural do transistor foi o surgimento do circuito integrado em 1958, criado por Robert Noyce. Um circuito integrado pode conter dezenas de transistores, executando desde funções lógicas simples até as funções mais complexas. A vantagem está no pequeno espaço ocupado, robustez a interferências elétricas e baixo consumo.

A pioneira no uso de circuitos integrados em computadores foi a IBM, que estabeleceu uma linha de computadores – o sistema 360 – para substituir os tipos 7094 e 1401 (veja a Figura 2.22).

O sistema 360 era vendido em diversos modelos para atender às exigências de custo e desempenho e foi lançado em 1965. Foi o primeiro computador de propósito geral produzido, atendendo tanto a processamento científico quanto a comercial, e as versões posteriores baseadas nessa arquitetura são utilizadas até hoje.

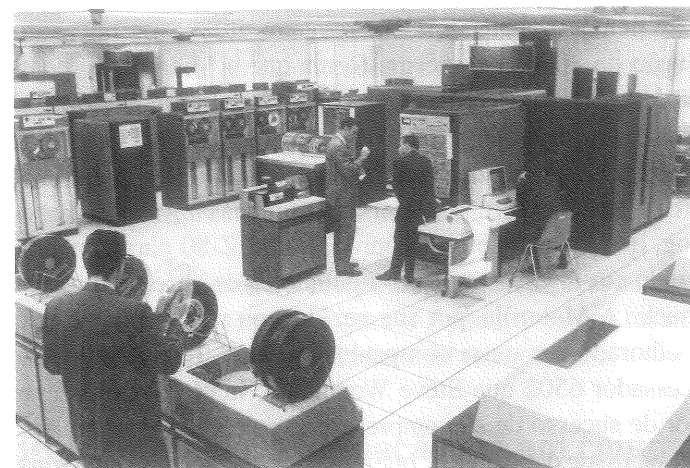



Figura 2.22 O computador IBM 360.

O primeiro minicomputador com circuitos integrados foi o PDP-11 da DEC, uma evolução do PDP-8. Essa máquina obteve um grande sucesso de vendas, sendo adotada principalmente por universidades.

2.2.5 Quarta geração – Computadores com *chips* VLSI

A quarta geração é marcada pelos microprocessadores. Um microprocessador é um dispositivo eletrônico encapsulado em um *chip* que possui internamente uma unidade de controle, uma unidade lógico-aritmética e uma memória interna, englobando as unidades funcionais básicas de um computador (leia a Seção 2.4). O primeiro microprocessador que surgiu foi o Intel 4004, de 1971, originalmente desenvolvido para uma empresa japonesa de calculadoras. O Intel 4004 era um microprocessador de 4 bits, isto é, poderia utilizar até 2^4 posições de memória.

Rapidamente a Intel percebeu que o 4004 poderia ser usado em outros projetos e então decidiu lançar um microprocessador mais poderoso, o 8008 de 1972. Foi com esse microprocessador que surgiu o primeiro microcomputador do mundo, na França, o *Micral* de 1973, que não obteve êxito. O Micral era programado diretamente com números binários, não possuindo nenhum periférico de entrada ou saída. As versões posteriores contavam com um software montador (*assembler*) desenvolvido por Philippe Kahn, que mais tarde fundaria a Borland International.

Com a popularização do microprocessador, surgiram diversos *kits* que podiam ser comprados em lojas e montados em casa. O mais famoso desses foi o *Altair 8800*,

que, em 1974, era vendido por US$ 439 e também utilizava o microprocessador Intel 8080, uma evolução do 8008. Da mesma forma que o Micral, não possuía periféricos de entrada, nem de saída, mas contou depois com um interpretador Basic, escrito por Bill Gates e Paul Allen, de uma pequena empresa (na época) denominada Microsoft. Surgiram, assim, os primeiros microcomputadores.

Outras empresas também começaram a construir seus microprocessadores. Em 1974 é fundada a Zilog, que lança o seu microprocessador Z-80 para concorrer com a Intel. O Z-80 foi utilizado em diversos microcomputadores de sucesso, por exemplo, o TRS-80 da Radio Shack. A Motorola, por sua vez, lançou nessa época o microprocessador 6500, que foi melhorado por outra companhia, a MOS Technology, lançando o 6502. Foi o microprocessador 6502 que Steve Wozniak e Steve Jobs utilizaram no primeiro microcomputador de sucesso de sua empresa, a Apple Computer: o Apple II de 1976. O Apple II foi o sucessor do Apple I, que em 1975 já usava um monitor de TV como dispositivo de saída, um teclado para efetuar a entrada de dados e uma unidade de cassete para armazenar programas e dados. Foi para o Apple II que foram escritos os primeiros softwares utilitários para um microcomputador: a planilha Visicalc e o editor de textos Wordstar.

Em 1981, a IBM decidiu investir em microcomputadores com o lançamento do IBM-PC (Personal Computer). Em vez de desenvolver todo o projeto, a IBM resolveu montá-lo a partir de peças e software fornecidos por terceiros e ainda disponibilizou todo o seu projeto para empresas interessadas nele. O primeiro PC era baseado no microprocessador Intel 8088, de 16 bits e velocidade de *clock* de 4.77 MHz. Possuía 16 K de memória RAM padrão, expansível até 256 K, um ou dois acionadores de disquete de 160 K e monitor opcional. O preço inicial era de US$ 1.565, correspondente a aproximadamente US$ 4 mil atualmente. A disponibilidade do projeto do PC fez com que outros fabricantes iniciassem a construção massiva de PCs, o que tornou esse equipamento popular e mais vendido até hoje, embora possua concorrentes à altura ou mesmo superiores.

A tecnologia VLSI (Very Large Scale Integration), que surgiu na década de 1980, permitiu que milhões de transistores pudessem ser encapsulados em uma única pastilha, também denominada *chip*. Dessa forma, foi possível criar pastilhas mais complexas e poderosas, reduzindo ainda mais o tamanho dos computadores e aumentando sua velocidade e processamento. Assim, a Intel ampliou sua família de microprocessadores, lançando depois do Intel 8088 os microprocessadores Intel 80286, Intel 386, Intel 486, Pentium, Pentium MMX, Pentium II, Pentium III e então o Pentium IV, sendo que o projeto deste último, após diversas atualizações e modificações, vem servindo de base para os *chips* atuais da Intel.

É claro que não se deve esquecer do outro grande concorrente do IBM-PC, o Apple Macintosh. A Apple, apesar de ter sido a pioneira na popularização dos microcomputadores, perdeu a liderança assim que a IBM abriu o projeto de seu IBM-PC. O Apple II original foi substituído pelo Macintosh em 1984, que, depois de sucessivas versões, tornou-se uma família de máquinas, abrangendo desde *notebooks* até máquinas poderosas para o trabalho em rede. A linha Macintosh atual da Apple também é baseada nos *chips* Intel.

A história do desenvolvimento do computador é incompleta sem a história do desenvolvimento do software. Reservou-se para o Apêndice A um resumo com os fatos históricos relevantes da história do computador e de seu software.

2.3 A representação da informação em um computador

2.3.1 A eletrônica digital do computador

Os circuitos eletrônicos de um computador moderno operam com sinais de dois níveis distintos ou *binário*. A razão de se utilizar circuitos que operam sob a forma binária e não decimal está no fato de que essa solução é simples e de baixo custo de implementação. Além disso, com circuitos digitais binários e utilizando-se dos resultados da *lógica booleana* (veremos sua aplicação em programação no Capítulo 3), é possível implementar em hardware qualquer tipo de função lógica, permitindo, assim, a construção de diversos tipos de circuitos empregados em um computador, como registradores, memórias, microcontroladores e o próprio microprocessador.

O ingrediente básico das pastilhas (*chips*) utilizadas no projeto dos circuitos do computador e que permite o chaveamento entre dois níveis lógicos é o *transistor*. Um transistor é um componente eletrônico criado a partir de um material *semicondutor*, isto é, possui a propriedade de conduzir a corrente elétrica somente após uma tensão conveniente ter sido aplicada em seus terminais. Dessa maneira, uma das aplicações do transistor é a de servir como uma chave "liga–desliga", conforme ilustrado na Figura 2.23.

Nessa figura, em (a) o transistor é representado por um dispositivo de três terminais, conhecidos como *coletor*, *base* e *emissor*. Ao se aplicar uma tensão alta (V_H) em sua base, o transistor permite a condução de corrente elétrica entre o coletor e o emissor, fechando o circuito e produzindo como saída (V_O) uma tensão próxima de zero. Por outro lado, se for aplicada em sua base uma tensão convenientemente baixa ($V_L < V_H$), permitirá a passagem de uma corrente elétrica muito baixa entre os terminais coletor-emissor que, para fins práticos, se comporta como um circuito aberto.

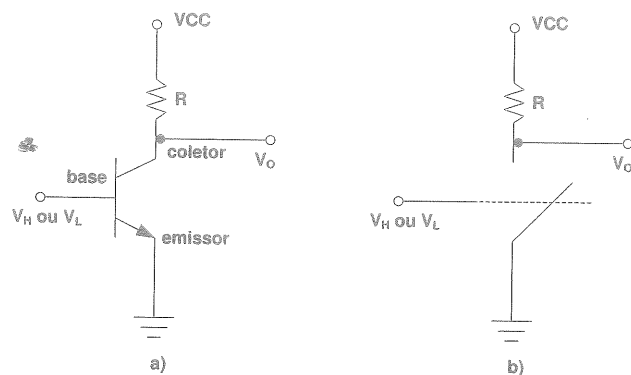



Figura 2.23 O transistor como chave.

Esse funcionamento pode ser descrito de forma simplificada como em (b). Nesse caso, o transistor é representado por uma chave controlada por tensão: se a tensão de entrada for alta, a resposta será um nível baixo; e se a tensão de entrada for baixa, a resposta será um nível alto. Convencionou-se chamar de nível alto o símbolo 1 e de nível baixo, 0. Esse exemplo ainda demonstra uma função lógica, o *inversor*: se a entrada for 1, a saída será 0; se a entrada for 0, a saída será 1.

Outras funções lógicas podem ser definidas da mesma forma, agrupando os transistores de forma conveniente. Não se intenciona aqui prolongar mais esse assunto. De qualquer maneira, lembre-se de que todos os dados armazenados e processados em um computador são traduzidos em sinais elétricos binários, ou seja, em um conjunto finito de 0s e 1s. Isso conduz ao conceito de *bit*.

2.3.2 Conceitos de bits e seus múltiplos

A palavra bit[2] ou *binary digit* representa de forma lógica um estado "ligado/desligado" ou *binário* existente em dispositivos eletrônicos digitais dos circuitos de um computador, como em registradores e memórias, por exemplo. Convenciona-se que um bit "ligado" é representado pelo símbolo 1 e "desligado", por zero.

O uso e a manipulação de números binários é similar ao dos números decimais. Aplica-se o mesmo conceito do sistema decimal, em que a posição de cada dígito de um número representa a potência da base (nesse caso, 2) que ele está figurado.

[2] A palavra bit, na realidade, surgiu na teoria matemática da informação, significando uma unidade básica de informação cuja incerteza é de 50%.

Por exemplo, o número 10101_2 binário pode ser entendido como[3]:

$$10101_2 = 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 21_{10}$$

Dessa forma, o número binário 10101_2 corresponde ao número decimal 21_{10}. De forma semelhante, pode-se converter um número decimal em binário, dividindo esse número sucessivamente por dois, até que o quociente seja zero. O número binário correspondente é obtido lendo-se os restos das divisões, da última para a primeira. Por exemplo, para converter o número 25_{10} em seu correspondente binário, utilizamos a seguinte sequência de operações:

$$\begin{array}{rrrrrr}
25 & \underline{|\ 2} & & & & \\
\underline{-24} & 12 & \underline{|\ 2} & & & \\
1 & \underline{-12} & 6 & \underline{|\ 2} & & \\
 & 0 & \underline{-6} & 3 & \underline{|\ 2} & \\
 & & 0 & \underline{-2} & 1 & \underline{|\ 2} \\
 & & & 1 & \underline{-0} & 0 \\
 & & & & 1 &
\end{array}$$

Assim, o número binário correspondente ao decimal 25_{10} é 11001_2 (confira).

Embora a unidade fundamental de informação do computador seja o bit, na prática utilizamos seus múltiplos, como o **byte**. Um byte representa o mesmo que oito bits e para fins de programação é o menor dado que se pode manipular diretamente. Os múltiplos do byte também são utilizados para representar as quantidades manipuladas e, geralmente, são o **quilobyte** (KB), o **megabyte** (MB) e o **gigabyte** (GB). Note que essas quantidades não são potências de dez e sim de dois, conforme ilustrado na Tabela 2.4.

Tabela 2.4 Os múltiplos do byte.

Nome	Valor
Quilobyte (KB)	1.024 (2^{10}) bytes
Megabyte (MB)	1.048.576 (2^{20}) bytes
Gigabyte (GB)	1.073.741.824 (2^{30}) bytes

Os tipos de informação manipulados pelo computador durante a execução de um programa são os dados e as instruções que operam sobre esses dados. Na memória são sempre representados por bits. Dentre os tipos de dados mais conhecidos temos

[3] Para separar números binários de decimais, se anotará um índice indicando qual é sua base, 2 ou 10.

os *caracteres*, as *cadeias de caracteres*, as *imagens* e os *sons* que serão discutidos nas seções a seguir. A representação binária das instruções é assunto da Seção 2.4.

2.3.3 Caracteres e cadeias de caracteres

Os caracteres são símbolos digitados pelo usuário durante a execução de seu programa ou que ainda podem ser constantes presentes no texto do programa. Envolvem as letras maiúsculas e minúsculas (A, B, C, . . ., Z, a, b, c, . . ., z), os números decimais (0, 1, 2, 3, 4, 5, 6, 7, 8, 9), símbolos especiais e de operação (+, -, *, /, SP (espaço), ", # etc.) e símbolos de controle (DEL (apagar), EOF (fim de arquivo), CR (retorno de carro) etc.) que não são visíveis na tela, mas que sinalizam alguma operação realizada no computador.

No texto, para se diferenciar um caractere de uma variável, o caractere será denotado pelo símbolo que o representa delimitado por *apóstrofes*, como, por exemplo, o caractere 'A', o caractere 'a', o caractere '1', e assim por diante.

É importante observar que o caractere '1' não representa o mesmo que o número inteiro 1. O anterior representa o símbolo '1', que foi, por exemplo, digitado pelo usuário em um programa enquanto o último representa um número inteiro, que pode ser manipulado em operações aritméticas.

Os caracteres são representados como números binários dentro do computador por meio de uma *tabela de caracteres*, que codifica os caracteres em números binários apropriados. Existem dois grandes sistemas para representar os conjuntos de caracteres: o EBCDIC (*Extended Binary Coded Decimal Interchange Code*) e o ASCII (*American Standard Code for Information Interchange*). O EBCDIC surgiu com o lançamento do computador IBM-360 na década de 60, e o ASCII foi definido para ser um padrão para a indústria de computadores e é o mais utilizado atualmente, em virtude de sua adoção pelos fabricantes de microcomputadores.

O conjunto de códigos ASCII original (128 símbolos) está descrito na Tabela 2.5. Para facilitar sua compreensão, os códigos dos caracteres estão representados apenas na base decimal.

Observe que, a partir dessa tabela, o código do caractere 'A' é 65, do caractere 'a', 97 e que as letras tanto maiúsculas quanto minúsculas estão representadas de forma contígua, isto é, o código de 'B' é 66, de 'b', 98 e assim por diante. Nessa tabela, percebe-se, também, a diferença entre o caractere '1' do número inteiro 1: o anterior vale 49_{10} (ou 110001_2) e o último é o próprio inteiro 1_{10} (ou 1_2).

As cadeias de caracteres representam uma sequência de caracteres que em um programa podem representar mensagens e textos. Uma cadeia de caracteres é também conhecida como *string*, que, traduzido do inglês, significa *sequência de elementos*. Na

memória do computador, uma cadeia de caracteres é representada pela sequência correspondente de seus códigos binários.

Tabela 2.5 A tabela de códigos ASCII.

Decimal	Caractere	Decimal	Caractere	Decimal	Caractere	Decimal	Caractere
0	NUL	32	␣	64	@	96	`
1	SOH	33	!	65	A	97	a
2	STX	34	"	66	B	98	b
3	ETX	35	#	67	C	99	c
4	EOT	36	$	68	D	100	d
5	ENQ	37	%	69	E	101	e
6	ACK	38	&	70	F	102	f
7	BEL	39	'	71	G	103	g
8	BS	40	(	72	H	104	h
9	HT	41	)	73	I	105	i
10	LF	42	*	74	J	106	j
11	VT	43	+	75	K	107	k
12	FF	44	,	76	L	108	l
13	CR	45	-	77	M	109	m
14	SO	46	.	78	N	110	n
15	SI	47	/	79	O	111	o
16	DLE	48	0	80	P	112	p
17	DC1	49	1	81	Q	113	q
18	DC2	50	2	82	R	114	r
19	DC3	51	3	83	S	115	s
20	DC4	52	4	84	T	116	t
21	NAK	53	5	85	U	117	u
22	SYN	54	6	86	V	118	v
23	ETB	55	7	87	W	119	w
24	CAN	56	8	88	X	120	x
25	EM	57	9	89	Y	121	y
26	SUB	58	:	90	Z	122	z
27	ESC	59	;	91	[	123	{
28	FS	60	<	92	\	124	|
29	GS	61	=	93	]	125	}
30	RS	62	>	94	^	126	~
31	US	63	?	95	_	127	DEL

Considere, por exemplo, a cadeia de caracteres:

```
A␣arte␣de␣programar.
```

Aqui o espaço em branco está escrito como ␣ para melhor visualização. Na memória do computador, essa cadeia de caracteres seria representada por números binários conforme a Tabela 2.6.

Tabela 2.6 Codificação de uma cadeia de caracteres.

Caractere	A	␣	a	r	t
Decimal	65	32	97	114	116
Binário	01000001	00100000	01100001	01110010	01110100

Caractere	e	␣	d	e	␣
Decimal	101	32	100	101	32
Binário	01100101	00100000	01100100	01100101	00100000

Caractere	p	r	o	g	r
Decimal	112	114	111	103	114
Binário	01110000	01110010	01101111	01100111	01110010

Caractere	a	m	a	r	.
Decimal	97	109	97	114	46
Binário	01100001	01101101	01100001	01110010	00101110

Note que a representação binária dos caracteres desse exemplo ocupa sempre oito bits. Embora com um conjunto de 127 elementos seja necessário somente sete bits para codificá-los[4] ($2^7 = 128$), pelo fato de que o byte é a menor unidade de informação que pode ser manipulada em um programa, cada caractere dessa tabela ocupa de fato um byte. A evolução da tabela ASCII original é a tabela **ASCII estendida**, possuindo 256 elementos e cada elemento ainda ocupa um byte[5]. Então, para fins práticos, se assumirá que um caractere ocupa um byte de memória.

2.3.4 Imagens

As imagens no computador são versões digitalizadas de imagens reais ou sintetizadas por algum software gráfico. Uma imagem, conforme percebida por nosso cérebro, é o resultado da interação das ondas eletromagnéticas componentes de uma fonte de luz refletida por uma imagem e que, ao alcançar células fotossensíveis denominadas *células-cone* existentes no fundo da retina, produzem e enviam sinais ao cérebro que então "formam" a imagem que se está vendo.

Existem três tipos de células-cone: aquelas que são sensíveis às frequências das cores

[4] A razão é que existem dois valores possíveis para ocupar cada posição de um bit, assim, com sete bits obtêm-se $2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2 = 2^7 = 128$ valores distintos.

[5] Isso não vale para esquemas de codificação tipo Unicode que propõem o uso de bytes múltiplos para representar qualquer caractere ou símbolo de qualquer língua do planeta.

verde, azul e vermelha. Dessa forma justifica-se um modelo de cor que é amplamente utilizado por artistas e também pela maior parte dos programas gráficos: o modelo RGB (RED-GREEN-BLUE). Nesse modelo, qualquer cor pode ser formada pela adição de intensidades adequadas de vermelho, verde e azul. Por esse motivo, essas três cores são denominadas *cores aditivas primárias*.

Voltando-se ao computador, o responsável pela geração das cores que são percebidas em um monitor de vídeo é o *subsistema de vídeo*. O subsistema de vídeo é normalmente uma placa "espetada" na placa-mãe do computador ou um circuito embutido na mesma e se conecta ao monitor de vídeo por um cabo de vídeo.

A placa de vídeo dita o *modo* de exibição que se obterá no monitor. No modo *texto*, as informações a serem exibidas são organizadas por desenhos de caracteres e dividem a tela em uma matriz ocupada tipicamente por 80 colunas e 25 linhas. Nesse modo, os programas exibem suas informações escrevendo os caracteres em posições dentro do espaço de 80×25 posições possíveis. Um exemplo de programa que utiliza esse modo é o programa *Edit*, um editor de textos distribuído juntamente ao sistema operacional Microsoft Windows e que é executado no ambiente MS-DOS, conforme a Figura 2.24.

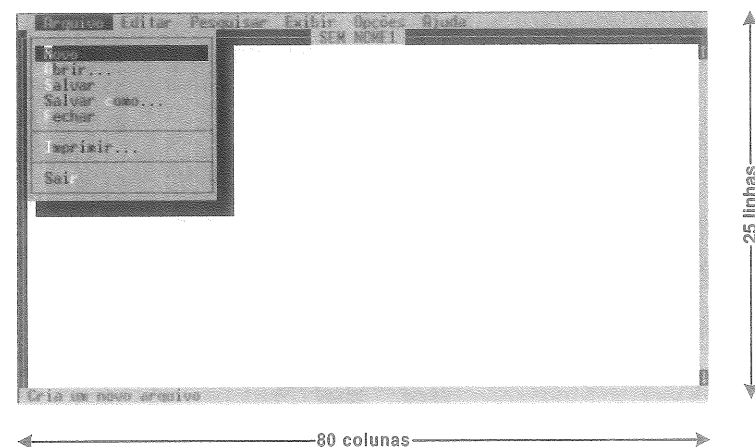



Figura 2.24 Tela em modo texto do programa *Edit*.

Cada caractere no modo texto é por si só representado por uma matriz de pontos – *pixels* ou *picture elements* –, que, organizados de forma conveniente, formam os caracteres que conhecemos. No modo texto, os atributos de cor para os caracteres são de apenas dois tipos: a cor de fundo (*background color*) e a cor do caractere (*foreground color*). Quando se alteram esses atributos de cor no modo texto, alteram-se a cor de todos os pixels correspondentes no caractere em questão. No modo texto, não é possível alterar a cor de um pixel individualmente.

Já no modo *gráfico*, a imagem percebida é formada por pixels que são individualmente acessados. Dessa forma, todas as informações visuais desenhadas na tela do computador são organizadas em uma grande matriz de pixels, cada um com sua própria cor. O tamanho dessa matriz depende da *resolução* da placa de vídeo. A resolução indica o número máximo de pixels que se visualizará, em linhas e colunas, e o produto desses números fornece a quantidade total de pixels que podem ser exibidos na tela.

A resolução de vídeo é uma propriedade da placa de vídeo que está instalada no computador. Assim, quando se fala em uma resolução de vídeo de 800×600 significa que a placa de vídeo em questão é capaz de exibir 800 vezes 600 pixels, ou 480.000 pixels. Um exemplo de uma tela em modo gráfico é representado pelas janelas dos aplicativos do sistema operacional Windows, nesse caso pelo programa de pintura *Paint*, conforme observado na Figura 2.25.

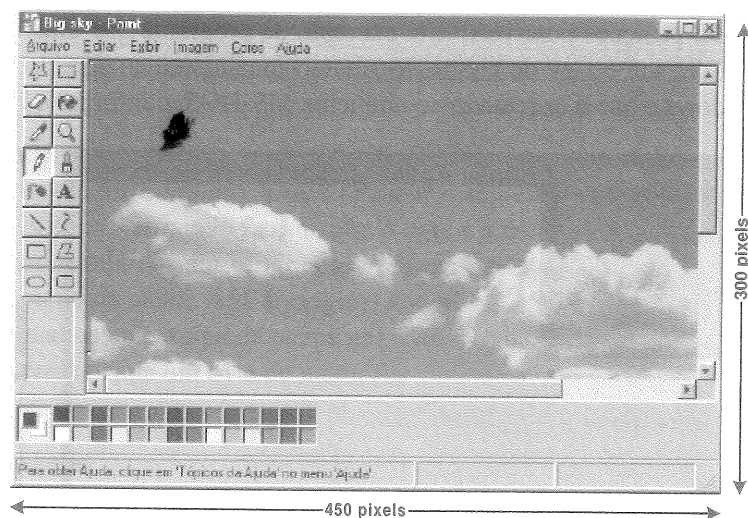



Figura 2.25 Tela em modo gráfico do aplicativo *Paint* do Windows.

No modo gráfico, emprega-se tipicamente a codificação da informação de cor, utilizando o modelo RGB já citado. Nesse caso, as cores possíveis são criadas a partir de quantidades de vermelho, verde e azul convenientes. Para o computador, esses números são representados por quantidades inteiras, no formato binário. O número total de cores que se pode apresentar no modo gráfico depende de um parâmetro do adaptador gráfico conhecido como *profundidade* de cor. A profundidade de cor é o número de bits utilizado para representar as cores oferecidas pelo adaptador gráfico.

Assim, ao comprar um adaptador gráfico, pode-se observar em sua especificação esse número. Atualmente é comum as placas com 32 bits de profundidade de cor, que são divididos nos matizes vermelho, verde e azul. Logo, uma cor de cada pixel no modo gráfico é representado por três números binários com oito bits cada (um byte, portanto), e cada um desses bytes representa a intensidade de vermelho, verde e azul na cor.

A quantidade de informação associada a uma imagem depende do seu tamanho em bits e da profundidade de cor utilizada. Uma imagem como a da Figura 2.25, se representada com uma informação de cor de 32 bits, vai ocupar $450 \times 350 \times 32/8$ bytes=1.920.000 bytes ou ainda 1.8 MBytes. Observe a necessidade de compressão de dados.

Portanto, seja no modo texto ou no modo gráfico, as informações desenhadas na tela do computador são representadas por dois atributos:

- Localização (coluna, linha do caractere em modo texto; coluna, linha do pixel em modo gráfico).
- Cor (atributos de fundo/frente em modo texto; cor RGB pixel em modo gráfico).

E essas informações deverão ser armazenadas como números binários na memória do computador, segundo algum esquema de codificação proposto pelo sistema operacional e dispositivo gráfico. Por exemplo, considere uma imagem de 4×4 pixels, conforme a Figura 2.26.

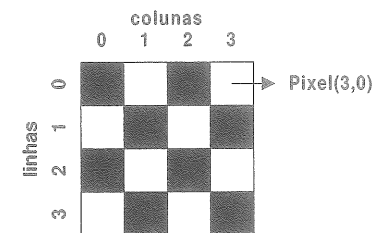



Figura 2.26 Exemplo de uma imagem.

Essa imagem contém somente duas cores: preto e branco. Presume-se que essas cores sejam representadas em uma placa com 32 bits de profundidade de cor e que possuam os seguintes códigos binários:

- Preto: 00000000000000000000000000000000.
- Branco:11111111111111111111111111111111.

Tabela 2.7 Codificação de uma imagem.

Pixel	Código
(0,0)	00000000000000000000000000000000
(0,1)	11111111111111111111111111111111
(0,2)	00000000000000000000000000000000
(0,3)	11111111111111111111111111111111
(1,0)	11111111111111111111111111111111
(1,1)	00000000000000000000000000000000
(1,2)	11111111111111111111111111111111
(1,3)	00000000000000000000000000000000
(2,0)	00000000000000000000000000000000
(2,1)	11111111111111111111111111111111
(2,2)	00000000000000000000000000000000
(2,3)	11111111111111111111111111111111
(3,0)	11111111111111111111111111111111
(3,1)	00000000000000000000000000000000
(3,2)	11111111111111111111111111111111
(3,3)	00000000000000000000000000000000

Na memória do computador, essas informações seriam armazenadas de forma contígua, como indicado na Tabela 2.7.

Nesse exemplo, considerou-se que as matrizes de dados foram armazenadas segundo o armazenamento contíguo de suas linhas (poderiam ser também por suas colunas).

2.3.5 Sons

Da mesma forma que as imagens, os sons são informações analógicas, contínuas no tempo e amplitude. A forma de onda de um som (música ou fala) pode ser considerada como a soma de diversas formas de onda senoidais, cada uma com uma *frequência* e *amplitude* particular. Um exemplo de um sinal de som analógico está na Figura 2.27.

O problema do armazenamento de informações de som no computador é análogo ao das imagens. Por ser uma forma de onda contínua, seu armazenamento na forma como é encontrada na natureza é inviável no computador. A solução encontrada e aplicada por todos os subsistemas de som do computador foi a *amostragem digital do sinal.*

O processo de amostragem digital de sinais analógicos, em particular o sinal de som, funciona da seguinte forma: o sinal de som é capturado via microfone ou saída de algum dispositivo analógico conectado à placa de som do computador. Em seguida, esse

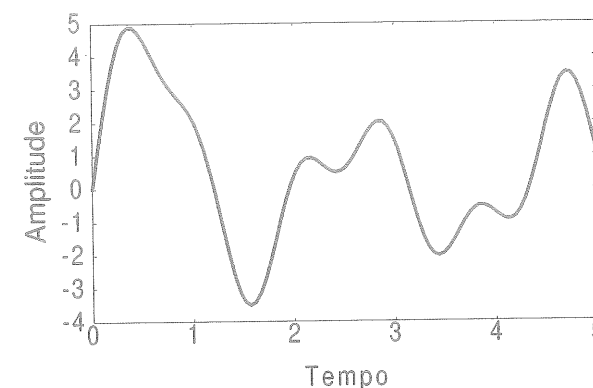



Figura 2.27 Exemplo de uma forma de onda de som.

sinal é *amostrado*, isto é, são coletadas amostras periódicas dele (sua amplitude). Isso é feito por um dispositivo conversor analógico-digital. Na sequência, a cada amplitude amostrada é atribuído um valor binário correspondente, sendo esse processo denominado *quantificação*. Por fim, esse sinal amostrado e quantificado é compactado e então armazenado de forma conveniente.

Por consequência desse processo de "discretização", a forma de onda resultante apresenta um "serrilhado", conforme exibido na Figura 2.28.

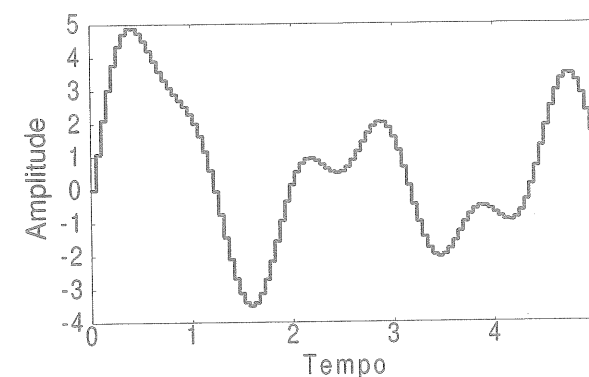



Figura 2.28 Exemplo de uma forma de onda de som após amostragem.

O processo de quantificação fornece os códigos binários para cada nível discreto. Um exemplo disso é exibido na Figura 2.29.

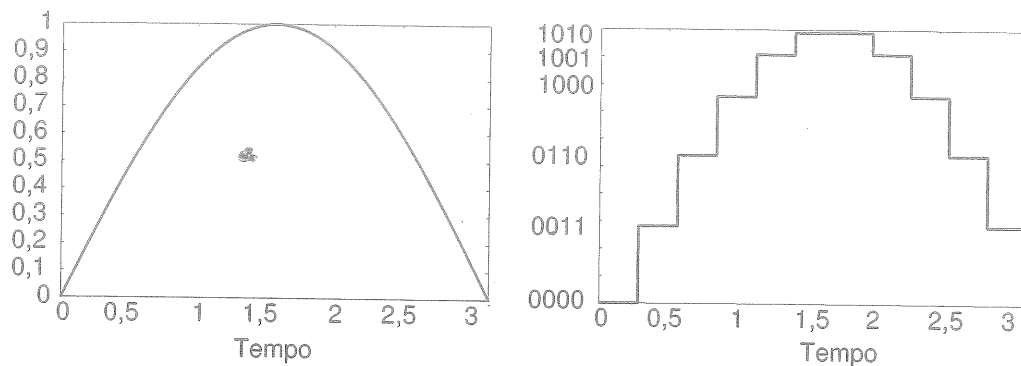



Figura 2.29 Exemplo de uma forma de onda de som após quantificação.

Nessa figura, um trecho ampliado da função $sen(x)$ no intervalo a é amostrado e quantificado por números de quatro bits a uma taxa de amostragem de três amostras por unidade de tempo. A representação desse sinal no computador poderia ser conforme a Tabela 2.8.

Tabela 2.8 Codificação de um sinal de som.

Amostra	Código
0	0000
1	0011
2	0110
3	1000
4	1001
5	1010
6	1010
7	1001
8	1000
9	0110
10	0011

Após ter sido armazenado, o som pode ser novamente reproduzido. Para tanto, aplica-se o caminho inverso: a partir das amostras existentes, o sinal é recomposto e filtrado para remover o efeito do "serrilhado". Por fim é enviado ao amplificador de saída da placa de som.

A qualidade do som a ser armazenado depende da frequência de amostragem empregada e da resolução da placa de som. Sabe-se pelo critério de Nyquist que a taxa de amostragem mínima para que o sinal possa ser recuperado novamente na forma analógica é de duas vezes a maior frequência existente no sinal original. Para a percepção do ouvido humano, uma taxa de amostragem de 44,1 kHz é suficiente para as falas e músicas.

Já a resolução da placa de som indica quantos níveis de quantificação são possíveis. Por exemplo, uma Sound Blaster 16 emprega uma resolução máxima de 16 bits, ou seja, 2^{16} níveis possíveis de quantificação. Um som amostrado com essa placa é, portanto, representado por números binários inteiros de 16 bits.

Um som de qualidade demanda espaço de armazenamento. Considerando uma taxa de amostragem de 44,1 kHz e uma resolução de 16 bits, um sinal de som de 60 segundos ocupará $44{,}1 \times 10^3 \times 60 \times 16$ bits, ou seja 42.336.000 bits ou 5 Mbytes! Da mesma forma que as imagens, os sons no computador devem ser comprimidos por algum padrão existente de compressão (por exemplo, MP3).

2.4 A arquitetura de um computador

A arquitetura de um computador representa a maneira na qual seus componentes estão organizados. Da história da computação (Seção 2.2), consagraram-se diversos modelos de arquitetura. Nesta seção será descrito o mais famoso e utilizado deles: o modelo de arquitetura de *Von Neumann*.

A *arquitetura de Von Neumann* surgiu em virtude de seu criador, John von Neumann, que, em 1946, definiu as unidades funcionais básicas para o projeto do computador IAS (veja a Seção 2.2.2). Essas unidades funcionais (ou componentes) estão ilustradas na Figura 2.30 e são:

- UCP (Unidade Central de Processamento) ou CPU (*Central Processing Unit*): é representada atualmente por um *microprocessador* e sua função é executar programas armazenados na *memória principal*. A memória principal ou ainda RAM, (*random access memory*), representa a memória volátil (apagada quando sem energia), utilizada para armazenar instruções e dados de um programa.

- Caminhos (*bus*) de dados: representam uma fiação (impressa na placa-mãe) e seu propósito é transmitir dados, que podem ser endereços de áreas da memória, dados de um programa armazenados na memória e sinais de controle para/de outros componentes externos, como, por exemplo, dispositivos de entrada/saída.

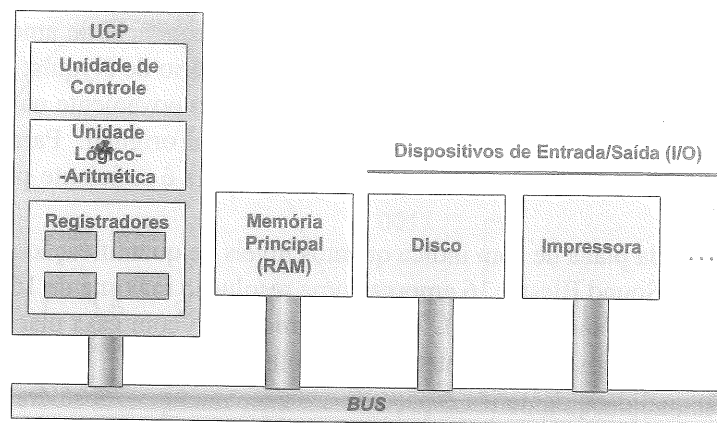



Figura 2.30 Organização típica de um computador.

- Dispositivos de entrada e saída ou dispositivos de I/O (*input/output*): representam interfaces para os dispositivos de entrada e saída, como, por exemplo, mouse, teclado, monitor, disco rígido, entre outros.

Esses componentes são representados por um conjunto de dispositivos eletrônicos digitais, encapsulados em *chips* (pastilhas) e em cartões de expansão que são respectivamente soldados e encaixados a uma placa especial denominada *placa-mãe* (ou ainda *motherboard* ou *mainboard*).

2.5 O funcionamento da UCP na execução dos programas

A UCP é a responsável pela execução dos programas e pelo controle dos outros componentes, como os periféricos de entrada e saída. A UCP é representada nos computadores pessoais pelo *microprocessador*. Simplificando, a UCP é internamente composta pelas seguintes unidades funcionais:

- uma unidade de controle (UC): responsável pela *busca* de instruções na memória principal;
- uma unidade lógico-aritmética (ULA): responsável pela execução de operações aritméticas e lógicas e pela *execução* das instruções provindas da memória principal;
- um conjunto de registradores: representa uma pequena memória (por exemplo, 32 registradores) que serve para armazenar resultados temporários e algumas informações de controle. Os registradores possuem normalmente um tamanho fixo em bits (por exemplo, 32 bits) e, por estarem situados internamente à UCP, são muito mais rápidos que a memória principal.

A organização interna de uma UCP é representada principalmente pelo seu *caminho de dados* (*data path*). Esse caminho indica o fluxo de dados que ocorre internamente na UCP. A Figura 2.31 exibe uma simplificação dessa organização.

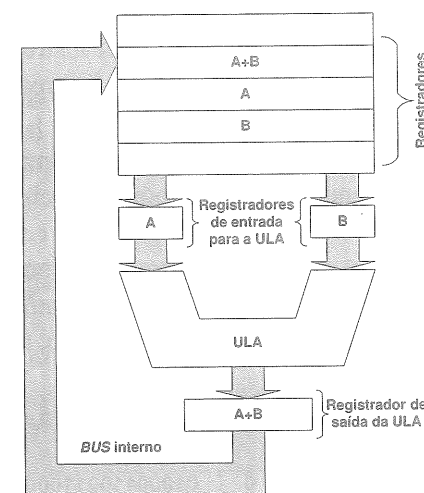



Figura 2.31 Caminho de dados de uma UCP.

O funcionamento do caminho de dados pode ser ilustrado com a operação de adição de dois números inteiros, representados simbolicamente pelas variáveis *A* e *B*. Supondo que esses dois números tenham sido armazenados em dois registradores da UCP, no momento da execução da instrução de soma, eles são transferidos para os registradores especiais da ULA, chamados de *registradores de entrada* (veja a Figura 2.31). Os circuitos internos da ULA entram em ação e realizam a soma desses números, sendo armazenados em outro registrador especial da ULA, denominado *registrador de saída*. Por fim, esse número resultante é armazenado em algum outro registrador da UCP, de acordo com a instrução de soma realizada.

Cabe aqui uma observação. Nota-se que a operação descrita não é realizada de uma única vez. Compete aos circuitos da ULA a correta execução dessas etapas de acordo com um *microcódigo* definido em seus circuitos que, por sua vez, é sincronizado por um sinal de *relógio*. Esse sinal de relógio é uma fração do relógio nominal do microprocessador. Por exemplo, um microprocessador com um relógio de 3,0 GHz significa que uma

onda quadrada com período de $1/(3,0 \times 10^9)$ segundos será utilizada como referência para os circuitos digitais do computador.

Esse ciclo de execução e posterior armazenamento do resultado é intitulado *ciclo de caminho de dados* e pode-se afirmar que quanto mais rápido esse ciclo ocorrer, mais rápido é o processador que o abriga.

As instruções que são executadas pela ULA dependem do conjunto de instruções definidos para a UCP em questão. Essas instruções em uma UCP típica devem abrigar pelo menos as dos tipos aritmética e lógica. Na arquitetura de Von Neumann, tanto as instruções quanto os dados necessários para essas instruções são armazenados na memória principal. Assim, surge a pergunta: como a UCP realiza o processo de busca e execução das instruções?

Primeiro, existe um registrador especial na UCP, denominado de *contador de programa* ou CP, que armazena o endereço da memória principal que contém a próxima instrução a ser buscada. A memória principal pode ser considerada como um conjunto de "caixinhas" em que são armazenados os dados ou as instruções (veja a Figura 2.32). Cada "caixinha" possui um endereço que a identifica. É claro que o número de "caixinhas" pode ser bem grande, podendo, por exemplo, armazenar 256 Mbytes.

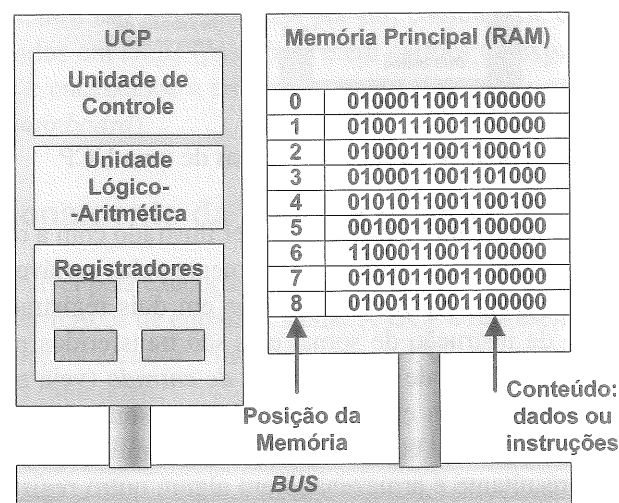



Figura 2.32 Memória principal do computador.

O funcionamento inicia-se da seguinte forma: a instrução armazenada no endereço apontado pelo registrador CP é buscada e armazenada em outro registrador especial, chamado *registrador de instrução* ou RI. Essa unidade de bits transferida da memória

para o registrador RI é nomeada como *palavra* de máquina. A largura da palavra de máquina depende da UCP, sendo típicos os valores de 8, 16, 32 e 64 bits.

Após a transferência da instrução para o registrador RI, o contador de programa (CP) é incrementado para apontar para a próxima instrução. A instrução presente no registrador RI é então decodificada para se saber qual o tipo de operação que representa. Se a instrução necessitar acessar um valor da memória, o endereço desse dado é armazenado em um registrador. Com o endereço armazenado em RI, o dado é buscado e armazenado em um registrador, que, por fim, é utilizado na execução da instrução propriamente dita.

A execução de um programa é, portanto, aquela repetitiva do parágrafo anterior até que uma instrução especial que indica o seu fim seja obtida. Esse funcionamento é resumido pelo Algoritmo 2.1.

Algoritmo 2.1 Algoritmo para o funcionamento da UCP.

Início
  $CP \leftarrow endereco_inicial$
  Enquanto $tipo_instrucao \neq PARE$ **Faça**
    $RI \leftarrow TransferirMemoria(CP)$
    $CP \leftarrow CP + 1$
    $tipo_instrucao \leftarrow DecodificarInstrucao(RI)$
    $endereco_dado \leftarrow PegarEnderecoDado(RI, tipo_instrucao)$
    Se $endereco_dado \geq 0$ **Então**
      $dado \leftarrow TransferirMemoria(endereco)$
    Fim Se
    $Executar(tipo_instrucao, dado)$
  Fim Enquanto
Fim

Falta ainda determinar como as instruções serão representadas. Essa é uma decisão do projetista da UCP. Por exemplo, em uma UCP com palavra de memória de 16 bits, as instruções poderiam ser representadas conforme a Figura 2.33. Nessa figura, o código da instrução é representado pelos bits 12 a 15 e o operando (endereço a ser utilizado pela instrução) pelos bits restantes. Assim, a instrução aritmética de soma poderia ser representada pelo código 0010 e assim por diante.

Figura 2.33 Exemplo de organização de instrução.

2.6 O projeto lógico na construção de programas

Por ser um texto introdutório de algoritmos e lógica de programação, a discussão a respeito de detalhes das instruções em nível de máquina termina por aqui. No entanto, deve ser explicado o processo de construção de um programa e reforçar o papel do estudo de algoritmos e lógica de programação nesse processo.

Um programa é para o computador um conjunto de instruções de máquina armazenadas na memória. Porém, normalmente essas instruções são geradas indiretamente, via arquivo texto contendo essas mesmas instruções em código de montagem (*assembly*), que são instruções mnemônicas, como ADD, MOV e outras mais fáceis de se lembrar que simples sequências de zeros e uns.

Mesmo assim, o uso de uma linguagem de montagem não é produtiva no sentido de criar programas em tempo hábil. Seu uso é destinado principalmente à programação de software de sistema (sistema operacional, por exemplo) e para softwares que operarão em tempo real, tais como drivers de vídeo, sistemas de controle industrial e outros.

Grande parte dos programas é na realidade escrita em linguagens de alto nível, que possuem instruções mais compreensíveis ao ser humano, como Pascal, Delphi, C, Java e C++. O processo de construção de um programa com essas linguagens segue as etapas ilustradas na Figura 2.34, que complementam as discussões feitas no Capítulo 1.

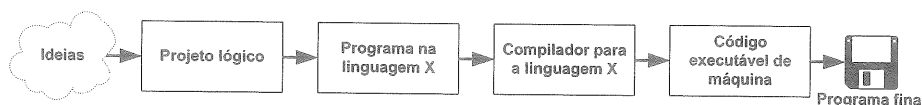



Figura 2.34 Etapas no desenvolvimento de um programa.

O processo de construção de um programa é iniciado pelas *ideias* que se tem a respeito do problema a ser resolvido. Depois, em uma etapa de planejamento, é realizado o *projeto lógico* do programa, assunto dos próximos capítulos deste livro. Essa etapa é crucial, pois é aqui que se definirá a lógica do programa em si e se ele servirá ou não como solução a um problema apresentado.

As etapas a seguir dependem da linguagem de programação que será utilizada. Considerando uma linguagem de programação *X*, a ideia nessa etapa é traduzir o projeto lógico para essa linguagem. Isso é feito com o conhecimento da equivalência das instruções definidas no projeto lógico com as instruções reais, disponibilizadas pela linguagem *X*.

Feito isso, o texto contendo as instruções do programa na linguagem *X* é submetido a um programa especial, denominado *compilador*. A tarefa do compilador é traduzir as instruções da linguagem *X* para aquelas de máquina do processador destino. O resultado é um programa executável (conhecido pela extensão .EXE no mundo DOS/Windows), que pode ser então colocado na memória pelo sistema operacional em questão e finalmente executado.

O projeto lógico, ponto de partida nesse processo, representa o programa em seu nível mais alto. Neste, os algoritmos que serão implementados são representados por gráficos, tais como fluxogramas, ou textos, como pseudolinguagem (como o Portugol), e são independentes de uma linguagem de programação. Daí se extraem algumas vantagens:

- Por serem independentes de uma linguagem de programação específica, os fluxogramas e o pseudocódigo podem ser reutilizados para definir programas que poderão ser implementados depois com qualquer linguagem de programação.
- O fluxograma e o pseudocódigo são ferramentas fáceis de aprender e mais fáceis de testar e verificar do que um programa escrito em uma linguagem de programação particular, pois não adicionam detalhes específicos dessas linguagens.
- Possuindo um projeto lógico verificado e testado, tornam-se mínimas as chances de escrever um programa com erros em uma linguagem de programação particular.

Capítulo 3

Algoritmos e Fluxogramas

No Capítulo 1 foi apresentado o conceito de algoritmo, suas características tais como a sua formalização via sintaxe e semântica adequadas e como o desenvolvimento de um algoritmo representa o desenvolvimento da solução de um problema. Neste capítulo será revisto o conceito de algoritmo para depois ser introduzida uma representação gráfica amplamente conhecida de algoritmos: fluxogramas.

3.1 Revisão do conceito de algoritmo

No Capítulo 1 houve um primeiro contato com a palavra algoritmo. Lá fazia-se a descrição de seu significado, de características desejáveis que todo algoritmo deve possuir, como sintaxe e semântica bem-definidas, e relacionava-se sua utilização com a solução de problemas. Para fixar o conceito de algoritmo, é fornecida a sua definição segundo o dicionário Aurélio:

> **Algoritmo**: 1. *Mat.* Processo de cálculo ou de resolução de um grupo de problemas semelhantes, em que se estipulam, com generalidade e sem restrições, regras formais para a obtenção do resultado ou da solução do problema. 2. *Inform.* Conjunto de regras e operações bem-definidas e ordenadas, destinadas à solução de um problema ou de uma classe de problemas, em um número finito de etapas.

Em outras palavras, o algoritmo representa o **caminho de solução para um problema**. A elaboração do algoritmo é de importância crucial para a criação de um programa de computador e nas soluções de qualquer tipo de problema. Da definição exposta anteriormente, pode-se extrair as seguintes características evidentes:

- Um algoritmo representa uma sequência de regras.
- Essas regras devem ser executadas em uma ordem preestabelecida.
- Cada algoritmo possui um conjunto finito de regras.
- Essas regras devem possuir um significado e ser formalizadas segundo alguma convenção.

3.2 Aplicabilidade dos algoritmos

Existe um algoritmo embutido em toda tarefa, independentemente de ela ser relacionada a um programa de computador. Em nosso cotidiano, executamos toda e qualquer tarefa utilizando algoritmos, mesmo não percebendo isso. Atos como comer, respirar, ir para a escola, dirigir um automóvel, resolver uma prova, estudar, cozinhar, fazer uma refeição, consertar o motor de um automóvel etc. são tarefas que podem ser descritas por meio de algoritmos.

Por outro lado, existem algoritmos que precisamos aprender para poder realizar certas tarefas específicas, como, por exemplo, aquelas ligadas à Engenharia e à Computação. Assim, para especificar um processo de montagem de um circuito eletrônico, um processo químico industrial e um programa eficiente de pesquisa de informações em um banco de dados, entre tantos, são necessários informações e conhecimentos adicionais aos que já possuímos. Desse modo, conclui-se que:

> *Algoritmos não servem apenas para programar computadores!*
> *São de uso geral!*

3.2.1 Exemplo não computacional de um algoritmo

Um exemplo concreto de um algoritmo que está fora do ambiente computacional é a receita para se preparar um sorvete de chocolate. Assim, o problema a ser resolvido é a definição dos passos necessários para se obter um sorvete de chocolate.

Aqui precisa-se saber inicialmente quais são os ingredientes essenciais para se fazer o sorvete. Uma sugestão seria utilizar os seguintes ingredientes:

- 1 tablete de chocolate meio amargo;
- 1 lata de leite condensado;

- a mesma medida da lata com leite;
- raspas de chocolate ou chocolate granulado.

Com esses ingredientes, especificam-se os passos da receita que resolve o problema da preparação do sorvete, representado pelo Algoritmo 3.1:

Algoritmo 3.1 Algoritmo para fazer um sorvete de chocolate.

Início
1. Ponha o chocolate em uma tigela refratária.
2. Deixe a tigela no micro-ondas durante um minuto em potência média.
3. Tire o chocolate do forno com cuidado e mexa-o até esfriar.
4. Bata-o no liquidificador com o leite condensado e o leite.
5. Despeje tudo em uma forma de gelo e espere congelar por três horas.
6. Distribua o sorvete em taças.
7. Decore com as raspas ou com o chocolate granulado.
8. Sirva.

Fim

3.2.2 Exemplo computacional de um algoritmo

Outro exemplo concreto, agora no domínio da Matemática, é o algoritmo de Euclides (definido entre 400-300 a.C.) para a determinação do *máximo divisor comum* entre dois números inteiros x e y. Os valores das variáveis x e y representam os valores de entrada do problema ou – fazendo uma comparação com o exemplo anterior – os "ingredientes". O algoritmo que resolve esse problema pode ser descrito conforme o Algoritmo 3.2.

Algoritmo 3.2 Algoritmo para calcular o máximo divisor comum entre dois números.

Início
1. Pedir para o usuário fornecer valores inteiros para x e y.
2. **Enquanto** $y \neq 0$ **Faça**
3. $r \leftarrow$ o resto da divisão entre x e y
4. $x \leftarrow y$
5. $y \leftarrow r$
6. **Fim Enquanto**
7. Exibir para o usuário o MDC procurado e que está em x.

Fim

Para verificar se esse algoritmo está correto, é necessário **simulá-lo** de acordo com suas regras. A linha 1 apresenta um comando que pede o fornecimento de dois valores inteiros, um para x e outro para y. Podem ser quaisquer valores, por exemplo, $x = 18$ e $y = 15$.

A linha 2 representa um comando que indica uma repetição, isto é, o que existe entre as linhas 2 e 6 deve ser repetido, enquanto a condição expressa na linha 2 ($y \neq 0$) for verdadeira. Dessa maneira, com os valores propostos, tem-se a geração de valores segundo a Tabela 3.1.

Tabela 3.1 Simulação do algoritmo de Euclides.

		Valor das variáveis		
Linha	**Comando**	x	y	r
1	Pedir para o usuário fornecer valores inteiros para x e y.	18	15	?
2	**Enquanto** $y \neq 0$ **Faça** (verdadeiro: $y = 15$)	18	15	?
3	$r \leftarrow$ o resto da divisão entre x e y	18	15	3
4	$x \leftarrow y$	15	15	3
5	$y \leftarrow r$	15	3	3
6	**Fim Enquanto**	15	3	3
2	**Enquanto** $y \neq 0$ **Faça** (verdadeiro: $y = 3$)	15	3	3
3	$r \leftarrow$ o resto da divisão entre x e y	15	3	0
4	$x \leftarrow y$	3	3	0
5	$y \leftarrow r$	3	0	0
6	**Fim Enquanto**	3	0	0
2	**Enquanto** $y \neq 0$ **Faça** (falso: $y = 0$)	3	0	0
7	Exibir para o usuário o MDC procurado e que está em x.	3	0	0

A Tabela 3.1 indica a execução dos comandos do algoritmo de Euclides, supondo valores iniciais como $x = 18$ e $y = 15$. Observe que a variável r nas duas primeiras linhas da tabela ainda não havia sido considerada pelo algoritmo, daí seu valor ser indeterminado. É importante notar que alguns comandos **alteram** o valor das variáveis e então novos valores passam a valer. Dessa forma, foi obtido como MDC entre x e y propostos o último valor que foi armazenado em x ($x = 3$).

Embora esses dois exemplos sejam algoritmos, existem ainda algumas deficiências em suas descrições. Entre elas:

- no primeiro exemplo consegue-se criar uma quantidade fixa de sorvete (essa receita rende seis taças). E se desejar obter somente uma taça? Não seria interessante conseguir uma solução mais geral, em que se possa variar as quantidades de ingredientes para obter a quantidade que quiser de sorvete? Lembre-se das soluções literais discutidas no Capítulo 1;

- no segundo caso, apesar de ser um algoritmo que determina o máximo divisor comum entre quaisquer valores inteiros de x e y, o algoritmo precisa ser formalizado. Nesse caso, é necessário definir uma sintaxe e semântica para representar esse algoritmo de uma maneira livre de interpretações ambíguas (a interpretação, por enquanto, está na tabela de simulação).

De qualquer modo, ambos os algoritmos apresentados possuem algumas propriedades em comum, que serão mais bem detalhadas na próxima seção.

3.3 Propriedades de um algoritmo

Todo algoritmo possui uma série de propriedades que serão descritas a seguir:

- **Valores de entrada**. Todo algoritmo deve possuir zero, uma ou mais entradas de dados. O exemplo do sorvete visto anteriormente na Seção 3.2.1 representa um algoritmo que possui zero entradas, pois seu algoritmo opera com quantidades fixas de ingredientes. Já o exemplo da Seção 3.2.2 representa um algoritmo com duas entradas de dados, atribuídas às variáveis x e y.

- **Valores de saída**. Todo algoritmo possui uma ou mais saídas, que simboliza(m) seu(s) resultado(s). Assim, tanto o algoritmo do sorvete quanto o de Euclides possuem uma única saída. No primeiro, a saída é o próprio sorvete; no segundo, o valor do máximo divisor comum entre x e y.

- **Finitude**. Costuma-se dizer que toda tarefa a ser realizada possui um início, meio e fim. Como os algoritmos representam os passos de solução de um problema – executando assim uma tarefa –, também possuem um início, meio e fim. Portanto, uma primeira propriedade do algoritmo é a finitude. Todo algoritmo deve ser finito, isto é, deve possuir um início e um conjunto de passos que, ao serem executados, levarão sempre ao seu término ou fim, executando a tarefa a que se propõe. Ambos os exemplos vistos nas Seções 3.2.1 e 3.2.2 são algoritmos finitos, pois chegam a um resultado em um número finito de passos.

  Deve-se uma atenção especial a essa propriedade. Muitas vezes, por desatenção, pode-se criar um algoritmo que nunca chegará a um resultado, tornando-se infinito. Por exemplo, altere a condição da linha 2 do Algoritmo 3.2 para $y \geq 0$.

Simule-o, então, para os valores $x = 5$ e $y = 2$ e responda: você consegue chegar à linha 7?

- **Passos elementares**. Um algoritmo computacional deve ser explicitado por meio de operações elementares, sem que possam haver diferenças de interpretação, de forma tal que possa ser executado até por *máquinas bastante limitadas*, como o computador.

  Dos exemplos vistos, o algoritmo de Euclides possui essa propriedade, pois utiliza somente operações matemáticas e comparações, operações que qualquer computador realiza por natureza. Já o algoritmo do sorvete deve ainda ser bem-refinado, para que suas operações possam ser representadas, de alguma maneira, em passos elementares.

- **Correção**. Um algoritmo deve ser correto, isto é, deve permitir que, com sua execução, se chegue à(s) saída(s) com resultados coerentes com a(s) entrada(s). Para saber se um algoritmo está correto ou não, deve-se realizar testes com diversos valores de entrada (simulação), cujos valores a serem produzidos já se conhece *a priori* e, então, comparar esses resultados com os valores produzidos pelo algoritmo em questão.

  Por exemplo: o máximo divisor entre os números 12 e 9 é 3. Faça $x = 12$ e $y = 9$ e então execute o algoritmo de Euclides mostrado na Seção 3.2.2. Você chegou ao mesmo resultado? Para outros pares de valores x e y, o algoritmo está correto?

3.4 Fluxogramas

Para um algoritmo ser útil, deve ser entendido da mesma forma por todas as pessoas que o utilizarem. Até o presente momento, as descrições de algoritmos que foram apresentadas usaram uma linguagem informal para representar os passos a serem executados. Apesar de cômodo, o uso de linguagens informais para a descrição de algoritmos pode levar ao surgimento de ambiguidades por diferentes pessoas.

Existem diversas maneiras de formalizar a representação de um algoritmo – neste livro utiliza-se uma forma de representação gráfica denominada fluxograma. A definição da palavra fluxograma, pelo dicionário Michaelis, é:

> **Fluxograma**: *Inform.* (fluxo+grama). 1. Diagrama para representação de um algoritmo. 2. Representação gráfica, por símbolos especiais, da definição, análise ou método de solução de um problema.

O uso de fluxogramas neste livro deve-se primeiramente ao fato de que o engenheiro deva possuir grande familiaridade com diagramas esquemáticos com linguagem matemática e expressão gráfica. Os fluxogramas possuem um grande apelo visual e aplicação no entendimento de processos industriais, os quais são muito importantes na formação e na vida prática do engenheiro. Além disso, a utilização de fluxogramas como elemento de representação na solução de problemas computacionais é ainda muito grande na ciência da computação.

Todo fluxograma deve possuir uma *sintaxe* e uma *semântica* bem-definidas. A sintaxe de um fluxograma é definida pela forma correta de empregar seus elementos, os quais são:

- símbolos gráficos específicos;
- expressões admissíveis a serem escritas no interior dos símbolos;
- sub-rotinas predefinidas que podem ser utilizadas nas expressões.

A semântica de um fluxograma indica como interpretá-lo. São regras de como entender e simular a solução que ele propõe. Tanto a sintaxe quanto a semântica de fluxogramas serão tratadas nas Seções 3.5, 3.6, 3.7, 3.8 e 3.9.

3.5 Construindo fluxogramas

Os símbolos de fluxograma a serem adotados neste livro seguirão a norma ISO 5807/1985. Uma descrição completa desses símbolos e de suas aplicações encontra-se no Apêndice B. Para se criar um fluxograma que represente um algoritmo, deve-se construir cada um de seus passos de acordo com um símbolo apropriado da norma. Serão apresentadas as regras básicas para a construção de fluxogramas nas seções a seguir.

3.5.1 Fluxograma mínimo

O menor fluxograma que se pode escrever é aquele que não executa absolutamente nada. De qualquer forma, todo fluxograma deve possuir um **início** e um **fim**. Os símbolos que denotam o início e fim de um fluxograma são representados por "retângulos arredondados", conhecidos por **terminadores**, contendo, respectivamente, os textos **Início** e **Fim**, conforme ilustrado na Figura 3.1.

Esses símbolos não representam nenhum tipo de operação, mas são essenciais para a determinação do início e fim do fluxograma. Os elementos de um fluxograma são conectados por **setas** que indicam o caminho a ser seguido a partir de um símbolo. A regra básica para a interpretação de um fluxograma pode ser sintetizada pelo Algoritmo 3.3.

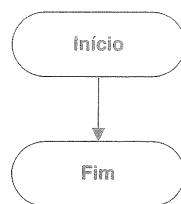



Figura 3.1 Fluxograma mínimo.

Algoritmo 3.3 Algoritmo para interpretar um fluxograma.

Início
1. Vá para o símbolo **Início**.
2. **Repita**
3. Seguindo a direção indicada pela seta, vá para o próximo símbolo.
4. Interprete esse símbolo.
5. **Até** chegar no símbolo **Fim**.

Fim

Assim, a interpretação do fluxograma da Figura 3.1 não leva à execução de nenhum comando, pois, após o símbolo **Início**, chega-se ao símbolo **Fim** sem passar por qualquer outro.

3.5.2 Fluxograma com comandos sequenciais

Um fluxograma contendo apenas comandos sequenciais é aquele que, a partir do símbolo **Início**, permite a execução das instruções contidas nos símbolos subsequentes sem desvio algum na direção até se alcançar o símbolo **Fim**.

Por exemplo, deseja-se construir um fluxograma que represente o algoritmo para calcular a força exercida pela coluna de um líquido sobre a área da válvula de um reservatório, conforme a Figura 3.2.

Nesse problema, conhece-se a altura h (m) do reservatório, o diâmetro d (m) da válvula e o peso específico γ do líquido (N/m^3). A força F, em Newtons, calculada é o peso da coluna do líquido. Como já temos o peso específico da substância, o cálculo da força é dado por:

$$F = \gamma \times VolumeColuna = \gamma \times AreaTampa \times h = \frac{\pi \times \gamma \times d^2 \times h}{4}$$

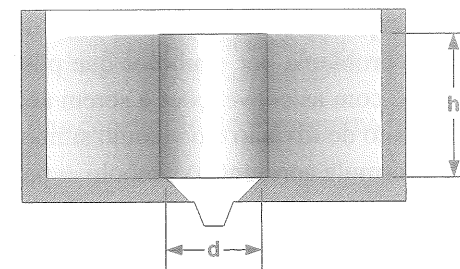



Figura 3.2 Problema da força exercida pela coluna de um líquido.

As variáveis existentes na fórmula anterior deverão também estar presentes na solução apresentada pelo fluxograma. No entanto, as **variáveis de um fluxograma** possuem um significado adicional daquele encontrado nas variáveis da Matemática: as variáveis em um fluxograma representam simbolicamente espaços da memória nos quais serão armazenados seus valores.

Deve-se ter em mente que uma variável, d, por exemplo, ao aparecer em um fluxograma, representa algum espaço da memória onde seu valor será armazenado, conforme indicado pela Figura 3.3. Não é necessário conhecer qual é a posição desse espaço na memória: sabendo-se apenas o nome da variável, pode-se **ler** ou **alterar** seu conteúdo.

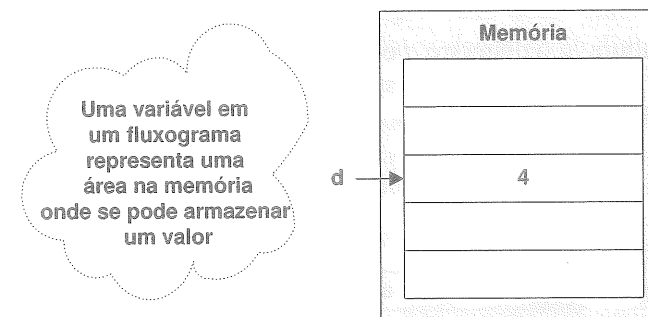



Figura 3.3 Significado de variável em fluxogramas.

Além disso, é necessário renomear as variáveis do problema original em alguns casos. Por exemplo, as variáveis π e γ desse problema serão escritas futuramente no texto de uma linguagem de programação, que contém apenas caracteres ASCII. Assim, seguindo as convenções que serão apresentadas na Seção 3.7, renomeiam-se as variáveis π e γ do problema, respectivamente, para *pi* e *gama*. Deve-se considerar, inclusive, que o símbolo *pi* é predefinido (veja a Seção 3.9) com o valor 3,14159.

Utilizando a mesma observação anterior, precisa-se substituir o símbolo de divisão e multiplicação da fórmula do problema por símbolos que possam ser escritos de uma forma mais simples. De acordo com as convenções a serem apresentadas na Seção 3.8, serão utilizados a / como símbolo de **divisão** e o $*$ como símbolo de **multiplicação**.

Observando, ainda, que a fórmula apresenta o símbolo $=$ para denotar que a variável F vai possuir o mesmo valor que o calculado pela expressão à sua direita, é necessário fazer outra observação. O símbolo $=$ será utilizado para realizar comparações, isto é, verificar igualdades. No caso desse problema, a variável F armazenará um valor calculado pela expressão. Dessa forma, para indicar essa operação, denominada **atribuição**, se utilizará o símbolo $\leftarrow$ para indicar que F armazenará o valor calculado.

Por fim, deve-se notar que na fórmula aparece o termo d^2. Para simplificar, será utilizada a sub-rotina predefinida *sqr*, que eleva ao quadrado o valor indicado em seu argumento (veja a Seção 3.9).

Observa-se que, para resolver esse problema de forma geral, devem-se obter, via algum **dispositivo de entrada** – por exemplo, um **teclado** –, os **valores de entrada** necessários para solucionar o problema. Essa operação é comumente chamada de **leitura dos dados**. De forma análoga, após a obtenção do resultado pela aplicação da fórmula, é preciso **exibir** o valor para o usuário, via algum **dispositivo de saída** – por exemplo, um **monitor de vídeo** –, o valor produzido, nomeado **valor de saída**.

Nesse problema, os valores de entrada e de saída são os seguintes:

- Entrada: a altura h, o diâmetro d e o peso específico $gama$.
- Saída: o valor da força F.

Agora já é possível esboçar um algoritmo informal para representar a solução desse problema, descrito pelo Algoritmo 3.4.

Algoritmo 3.4 Algoritmo para calcular a força exercida pela coluna de um líquido.

Início
1. Ler os valores h, d e $gama$.
2. Calcular $F \leftarrow pi * gama * sqr(d) * h/4$.
3. Exibir o valor de F.

Fim

Nesse instante se inicia a construção do fluxograma. Da forma que foi apresentado na Seção 3.5.1, a construção desse fluxograma começa com o desenho do símbolo de **Início**, conforme ilustrado na Figura 3.4.

Figura 3.4 Passo 1 na construção do fluxograma para o problema da força.

Na sequência, conecta-se esse símbolo ao primeiro passo a ser executado, usando-se uma seta. O primeiro passo, de acordo com o Algoritmo 3.4, é a leitura dos valores das variáveis h, d e $gama$. O símbolo do fluxograma para se ler os valores externos a serem atribuídos a variáveis do algoritmo tem a forma de um trapézio. Indica-se no seu interior os nomes das variáveis que receberão os valores a serem digitados, separados por vírgula, de acordo com a Figura 3.5.

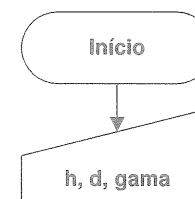



Figura 3.5 Passo 2 na construção do fluxograma para o problema da força.

Deve-se fazer duas observações sobre o significado desse símbolo. Em primeiro lugar, esse símbolo não indica qual tipo de tela os dados serão digitados. Isso reflete a característica de independência dos fluxogramas. Portanto, é de "imaginação livre" como esse comando funcionaria na prática. Por exemplo, ele poderia ser futuramente implementado em Pascal ou em Delphi e ser apresentado em telas como mostra a Figura 3.6.

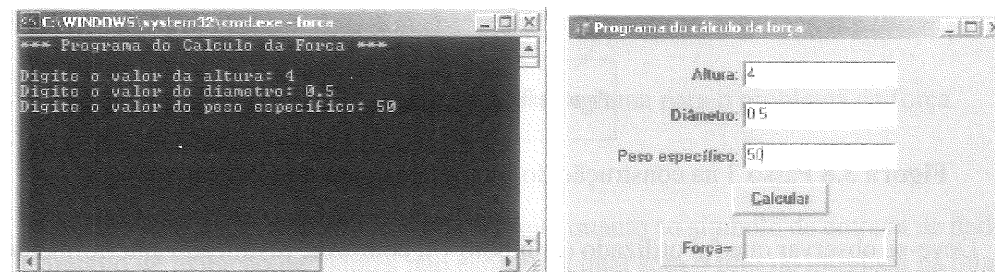



Figura 3.6 Duas interpretações concretas do símbolo de entrada.

Em segundo lugar, a execução do símbolo de entrada realiza uma **atribuição im-**

plícita dos valores digitados às variáveis que estão indicadas no seu interior, na ordem que foram escritas. Dessa forma, a execução do símbolo de entrada apresentado na Figura 3.5 com os valores 4, 0,5 e 50 vai atribuir esses valores, respectivamente, às variáveis h, d e $gama$, isto é, serão armazenados nas posições de memória reservadas para essas variáveis, conforme ilustrado pela Figura 3.7.

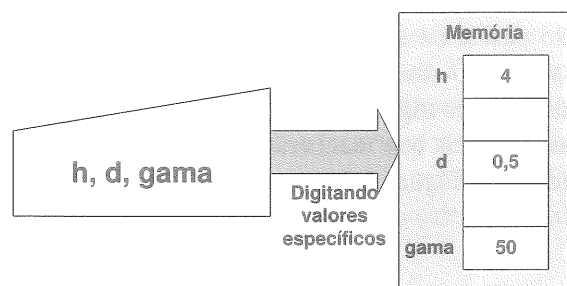



Figura 3.7 O efeito da entrada de dados nas variáveis.

Seguindo com a elaboração do fluxograma, o próximo passo representa um cálculo direto, portanto, **um processo**. O símbolo de fluxograma para representar cálculos ou processos a serem **executados sem questionamento** tem a forma de um retângulo. Em seu interior, escrevem-se as expressões ou o nome do processo que será executado. Nesse exemplo, calcula-se a força F, de acordo com a Figura 3.8.

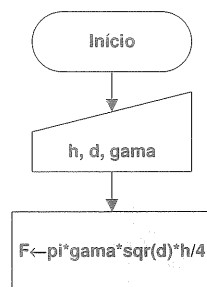



Figura 3.8 Passo 3 na construção do fluxograma para o problema da força.

Deve-se observar que foi utilizado o símbolo ←, conforme justificado anteriormente para **atribuir** o valor do cálculo realizado pela expressão à variável F. Assim, com os valores $h = 4$, $d = 0{,}5$ e $gama = 50$, será armazenado o valor 32,2699 na posição de memória que a variável F representa, como mostra a Figura 3.9.

No próximo passo, deve-se exibir o valor calculado da variável F. O símbolo do

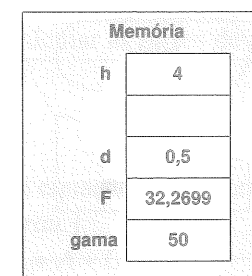



Figura 3.9 O efeito do comando de atribuição em uma variável.

fluxograma para **exibir** os valores está representado a seguir e, em seu interior, escrevem-se os nomes das variáveis a serem exibidas separados por vírgula. Nesse caso, deve-se exibir apenas o valor de F, conforme indicado na Figura 3.10.

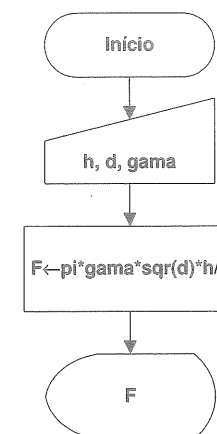



Figura 3.10 Passo 4 na construção do fluxograma para o problema da força.

Aqui, vale a mesma observação que foi feita quanto ao símbolo de entrada de dados. O símbolo de exibição de valores não especifica qual será o tipo de interface na qual o dado aparecerá (veja novamente a Figura 3.6).

Por fim, é necessário indicar o término do fluxograma por seu símbolo **Fim**. O fluxograma final é apresentado na Figura 3.11.

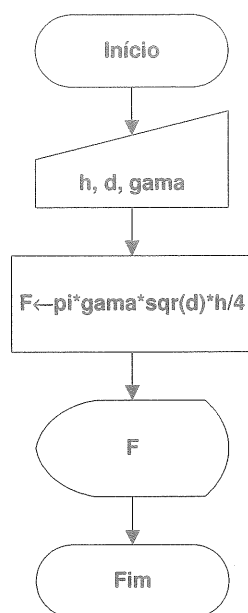



Figura 3.11 Fluxograma final para o problema da força.

3.5.3 Fluxograma com comandos de decisão

Deseja-se criar um fluxograma que represente o algoritmo para calcular as raízes de uma equação de segundo grau tipo $Ax^2 + Bx + C$, utilizando a fórmula de Bhaskara a seguir:

$$x = \frac{-B \pm \sqrt{B^2 - 4AC}}{2A}$$

Esse algoritmo deverá gerar uma resposta para quaisquer valores de A, B e C fornecidos, como mostra o Algoritmo 3.5.

Nesse algoritmo foi usada a rotina predefinida $sqrt$ para representar a operação raiz quadrada (veja a Seção 3.9).

Observe com atenção a linha 2 do Algoritmo 3.5. Nota-se que é utilizado um comando de decisão. Nesse caso, é necessário verificar se $A \geq 0$ e, dependendo do resultado, tomar uma das seguintes decisões (mutualmente exclusivas):

- se for verdade, exiba mensagem "Não é equação de 2º grau" e então pare;
- senão, continuar em frente com os cálculos.

Algoritmo 3.5 Algoritmo para calcular as raízes de uma equação de 2º grau.

Início
1. Ler A, B, C.
2. **Se** $A = 0$ **Então**
3. Exiba a mensagem "Não é equação de 2º grau!".
4. **Senão** {*A equação é de 2º grau.*}
5. Calcule $D \leftarrow sqr(B) - 4 * A * C$
6. **Se** $D < 0$ **Então**
7. Exiba a mensagem "Não existem raízes reais!".
8. **Senão** {*Calcule as raízes.*}
9. $r1 \leftarrow (-B + sqrt(D))/(2 * A)$
10. $r2 \leftarrow (-B - sqrt(D))/(2 * A)$
11. Exibir $r1$ e $r2$
12. **Fim Se**
13. **Fim Se**

Fim

Repetindo o mesmo que na Seção 3.5.2, o início do fluxograma fica conforme a Figura 3.12.

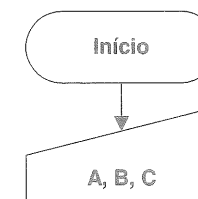



Figura 3.12 Passo 1 na construção do fluxograma para o problema das raízes.

A norma ISO 5807/1985 possui um símbolo específico para realizar esse tipo de operação – chamada **decisão** – que possibilita escolher um de dois caminhos a partir de um teste. Seu símbolo possui a forma de um losango, no interior do qual se escreve a expressão do teste a ser avaliado. Desse símbolo partem duas setas que, dependendo do valor da expressão, indicarão o caminho a ser seguido.

O resultado dessa expressão é um valor **lógico**, considerando um de dois valores possíveis: **verdadeiro** ou **falso**. Convencionamos escrever esses valores, respectivamente, pelas palavras em inglês ***true*** e ***false*** (veja a Seção 3.6).

Dessa forma, até a linha 2, o fluxograma ficaria de acordo com a Figura 3.13.

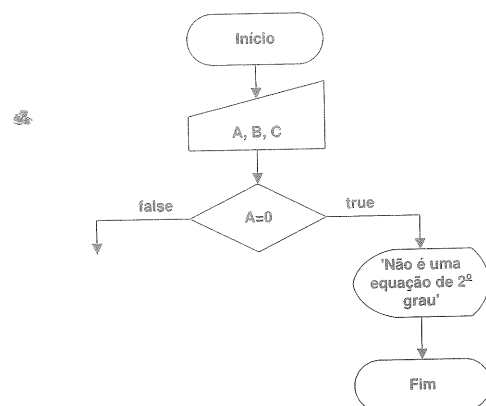



Figura 3.13 Passo 2 na construção do fluxograma para o problema das raízes.

Indicam-se nas retas que saem do símbolo de decisão os dois valores possíveis de um teste ou comparação (*true* e *false*), sendo, por convenção, *true* no lado direito e *false* no lado esquerdo. Mensagens constantes (cadeias de caracteres fixas) são delimitadas por apóstrofes (veja a Seção 3.6), para evitar que sejam confundidas com as variáveis.

O caminho a ser tomado após a avaliação da expressão lógica em um símbolo de decisão depende do resultado dessa expressão. A Figura 3.14 exibe dois casos que, dependendo do valor da variável A, vão conduzir a caminhos distintos.

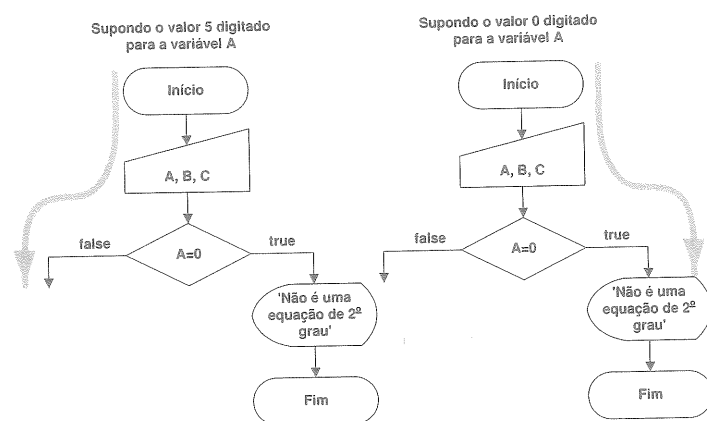



Figura 3.14 Encaminhamento após um comando de decisão.

O comando da linha 5 do Algoritmo 3.5 só deverá ser executado se o valor da variável A não for zero. Logo, o fluxograma fica conforme a Figura 3.15.

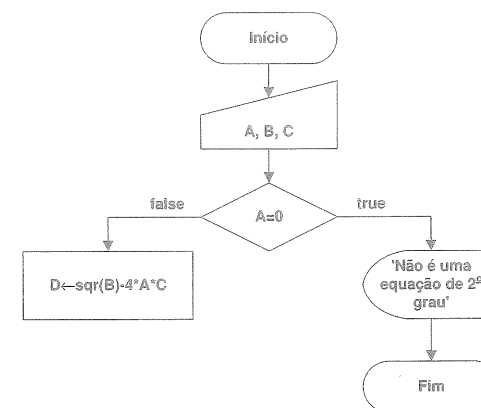



Figura 3.15 Passo 3 na construção do fluxograma para o problema das raízes.

Por fim, o comando da linha 6 do Algoritmo 3.5 introduz outra estrutura condicional. Seguindo o raciocínio análogo ao feito anteriormente, completa-se o fluxograma. Sua versão final está representada na Figura 3.16. Observe o uso do símbolo representado por uma "bolinha". Ela não representa comando algum, mas uma forma de "juntar" os fluxos que provêm de caminhos diferentes.

Nota-se que na Figura 3.16 foram adicionadas anotações nas laterais dos símbolos utilizados, representando **comentários**. Não executam comando algum e servem para tornar mais claras as partes do fluxograma para o leitor. Os comentários tornam mais rápido o entendimento de um fluxograma, porém aqueles que forem óbvios devem ser evitados.

3.5.4 Fluxograma com comandos de repetição

Nesta seção serão apresentados os fluxogramas que contêm *comandos de repetição*. Para tanto, será utilizado como exemplo o algoritmo de Euclides para o cálculo de máximo divisor comum entre dois números inteiros, segundo o Algoritmo 3.2.

Antes de iniciar a construção do fluxograma propriamente dito, devem ser feitas algumas observações. Observe que na linha 2 desse algoritmo aparece a expressão $y \neq 0$. Seguindo as convenções que serão apresentadas na Seção 3.8, o símbolo para representar a **desigualdade** será o $<>$ (justaposição dos símbolos $<$ e $>$). Já na linha 3, deve

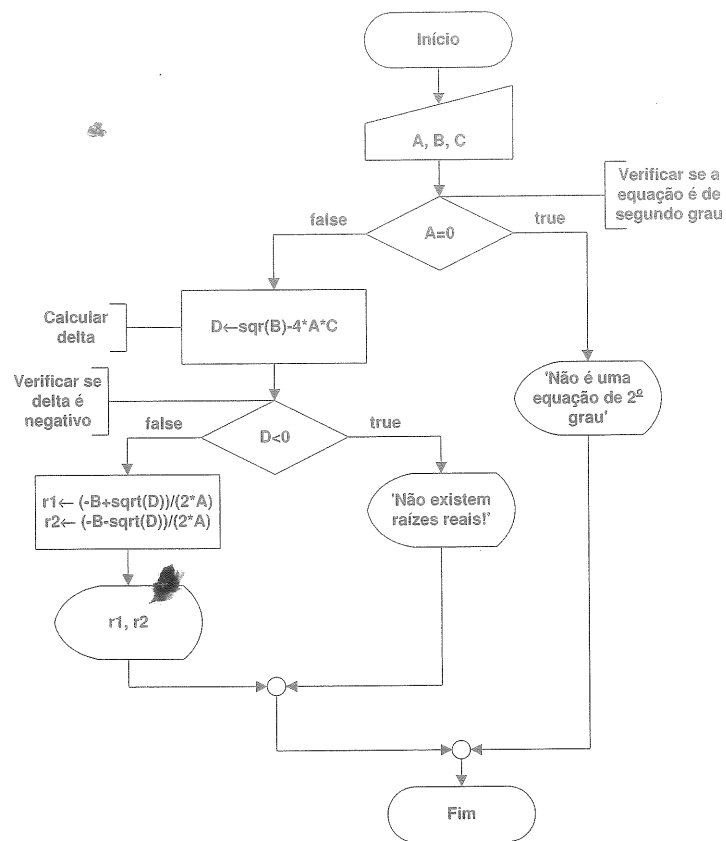



Figura 3.16 Fluxograma final, comentado, para o problema das raízes.

ser executado o cálculo do **resto da divisão entre dois números inteiros**. Acompanhando as convenções que serão apresentadas na Seção 3.8, o símbolo de operador para representar essa operação será **mod**.

O fluxograma para representar esse algoritmo está na Figura 3.17. Devem ser feitos alguns comentários:

1. Embora seja empregado nesta seção o mesmo símbolo do comando de decisão da norma ISO 5807/1985, aqui ele possui um significado diferente: é parte integral de um comando de repetição, servindo como um teste para indicar se os comandos em seu interior deverão ser executados novamente.

2. A repetição é indicada pelo caminho fechado que sai do símbolo de decisão e que

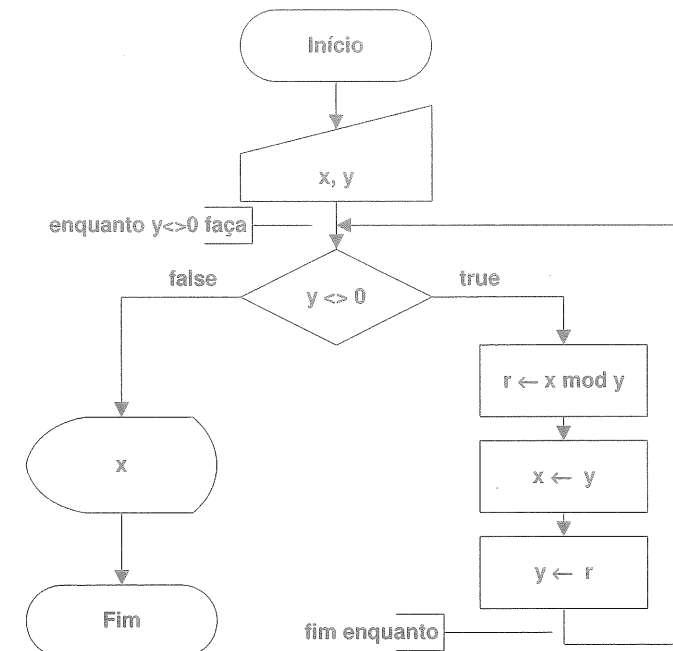



Figura 3.17 Fluxograma para o algoritmo de Euclides.

volta para ele.

3. A linha de retorno deve ser sempre desenhada imediatamente antes do símbolo de decisão.

4. A expressão lógica escrita internamente ao símbolo de decisão, no caso de comandos de repetição, representa um **critério ou condição de parada** da repetição. No caso desse fluxograma, os símbolos desenhados no caminho fechado representam os comandos que deverão ser executados **enquanto** a condição $y <> 0$ for verdadeira. O bloco que exibe o resultado contido na variável x apenas será executado quando a condição $y <> 0$ for falsa. Esse comando de repetição será formalizado no Capítulo 4 com o nome de estrutura **enquanto-faça**.

5. As expressões de condição de parada devem ser testadas, pois, se estiverem erradas poderão levar a uma **repetição infinita**. Por exemplo, se for trocada a condição $y <> 0$ para $y >= 0$ no fluxograma apresentado na Figura 3.17, nunca se alcançará o símbolo **Fim**, tornando-o infinito, portanto, deixando de representar um algoritmo.

Não existe somente essa forma de escrever um comando de repetição. De fato, é possível posicionar a condição de parada após a execução dos comandos a serem repetidos, conforme ilustrado na Figura 3.18. Cabem aqui mais algumas observações:

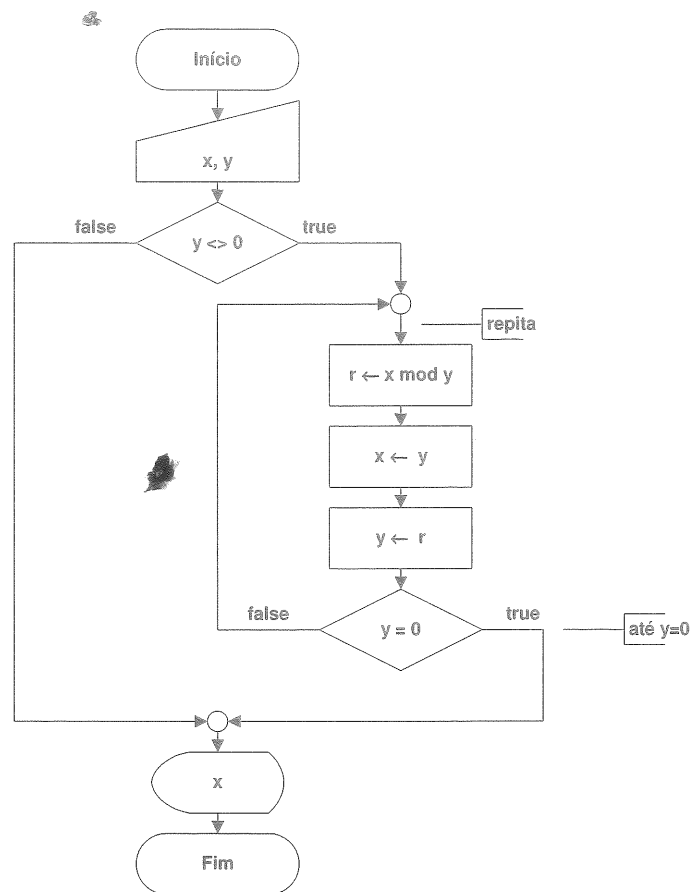



Figura 3.18 Outro fluxograma para o algoritmo de Euclides.

1. Note que o comando de repetição foi escrito agora com a condição de parada *após* os comandos que se desejam repetir. Esse comando pode ser entendido como "**repita** os comandos **até que** a condição de parada seja verdadeira". Com efeito, esse comando será formalizado em uma estrutura de programação denominada **repita-até**, a ser apresentada no Capítulo 4.

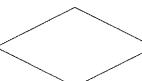
2. Como esse comando de repetição primeiro executa os comandos para depois verificar a condição de parada, foi necessário incluir um teste ($y <> 0$) antes dessa repetição, pois, caso contrário, existe a possibilidade da realização de uma divisão por zero (no comando $r \leftarrow x \ mod \ y$).

3. A condição de parada mudou. Nesse caso, a repetição será executada se a condição $y = 0$ for falsa, que, em termos lógicos, é o mesmo que executar a repetição se a condição $y <> 0$ for verdadeira. Nota-se, ainda, que a repetição termina quando a condição $y = 0$ é verdadeira.

Os símbolos vistos neste capítulo até este ponto estão resumidos na Tabela 3.2.

3.5.5 Simulação de algoritmos com fluxogramas

A simulação de um fluxograma é feita da mesma maneira que a de um algoritmo (veja a Seção 3.2.2). A vantagem é que os símbolos possuem um significado preciso e as setas que conectam esses símbolos permitem seguir o fluxo das instruções de uma forma mais visível. Uma sequência para se entender e simular um fluxograma foi definido pelo Algoritmo 3.3.

Como exemplo final desta seção, será projetado e testado um fluxograma para resolver o seguinte problema: exibir a média de N temperaturas fornecidas pelo usuário.

Deve-se entender o problema, com a experiência pessoal obtida ou por meio de dados fornecidos. Os dados do problema induzem:

1. O número de temperaturas é variável e indeterminado, representado simbolicamente pela variável N.

2. Para realizar a média de N temperaturas é necessário, primeiramente, que esses N valores sejam fornecidos pelo usuário.

3. Não é possível resolver esse problema se $N \leq 0$, pois não tem sentido fazer a média de um conjunto com zero ou menos valores.

4. Com esses valores, agora é possível calcular a média: basta somar todos os N valores e depois dividir o resultado por N.

A primeira questão que surge é a seguinte: se o número N de valores de temperatura for indeterminado e se para cada temperatura for necessária uma variável para armazená-la, como resolver o problema?

Na realidade, a solução é bem simples. Esse problema não exige que se armazenem todas as variáveis de temperatura (como armazenar conjuntos variáveis de dados será

Tabela 3.2 Resumo dos símbolos vistos no Capítulo 3.

Símbolo	Nome	Utilidade
	Terminador	Representar a saída para ou entrada do ambiente externo, por exemplo, início ou final de programa, uso externo e origem ou destino de dados etc.
	Processo	Representar qualquer tipo de processo, processamento de função, por exemplo, executando uma operação definida ou grupo de operações, resultando na mudança de valor, forma ou localização de uma informação ou determinação de uma, entre as várias direções de fluxo a serem seguidas.
	Linha básica	Representar o fluxo dos dados ou controles. Podem ser utilizadas pontas de seta, sólidas ou abertas, na extremidade para indicar a direção do fluxo onde necessário ou para enfatizá-lo e facilitar a legibilidade.
	Entrada manual	Representar os dados, de qualquer tipo de mídia, que sejam fornecidos, manualmente, em tempo de processamento, por exemplo, teclado *on-line*, mouse, chaveamento, caneta óptica *light pen*, leitor de código de barras etc.
	Exibição	Representar os dados, cuja mídia seja de qualquer tipo, na qual a informação seja mostrada para uso humano, tais como monitores de vídeo, indicadores *on-line*, mostradores etc.
	Decisão	Representar uma decisão ou um desvio tendo uma entrada; porém pode ter uma série de saídas alternativas, uma única das quais deverá ser ativada como consequência da avaliação das condições internas ao símbolo. O resultado apropriado de cada saída deverá ser escrito adjacente à linha, representando o caminho respectivo.

visto no Capítulo 5). Particularizando o problema, se N fosse igual a um, o problema seria resolvido de forma direta, bastando apenas exibir a única temperatura digitada.

No entanto, existem N temperaturas a considerar e sabe-se que o problema somente será resolvido de forma correta para os valores de N maior que zero. Como não se deve confiar nos valores que o usuário digita, uma primeira versão do fluxograma que contemple a solução desse problema deverá permitir a entrada do valor N e então decidir se N é maior que zero ou não. No caso negativo, basta exibir uma mensagem de erro adequada e, então, terminar, conforme ilustrado pela Figura 3.19.

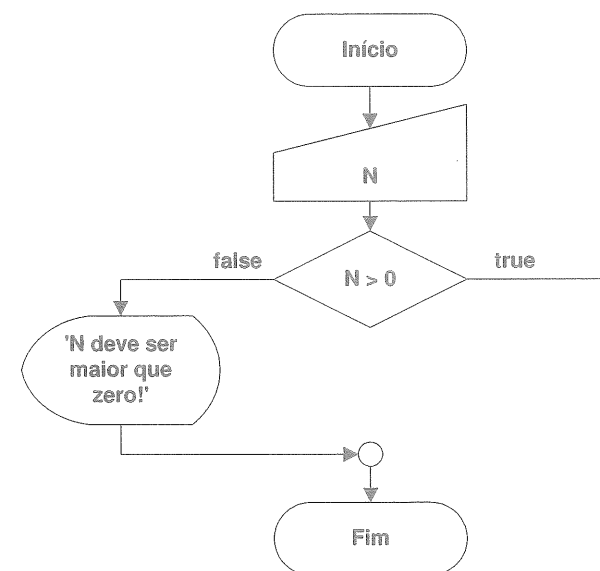



Figura 3.19 Passo 1 na construção do fluxograma para o problema das temperaturas.

Agora surge o ponto crucial: como ler, somar e realizar a média de todas as temperaturas? Basta pensar na utilização de um comando de repetição dentro do qual será lido um valor de temperatura e, a seguir, acrescentá-lo aos outros que já foram somados. Quando se somarem todos os valores, basta realizar a divisão por N.

Conduzindo a solução por partes, primeiro é preciso saber como controlar o número de repetições. Deseja-se que, para um valor $N > 0$, sejam repetidas a leitura e a soma de valores. Para criar um comando de repetição que vai executar N repetições, é necessária uma variável adicional, denominada comumente *variável contadora*, ou simplesmente, *contador*.

A ideia é fazer que essa variável comece com um valor inicial adequado e então repetir a instrução de incremento dela até que ultrapasse o valor final, que, nesse caso, é N. Se o nome da variável contadora for i, faz-se inicialmente que i tenha o valor 1 e, realizando incrementos unitários nessa variável, após N repetições, ela alcançará um valor maior que N. A condição que permitirá a realização das repetições pode ser escrita como $i \leq N$ e servir, também, como critério de parada assim que $i > N$, conforme descrito na Figura 3.20.

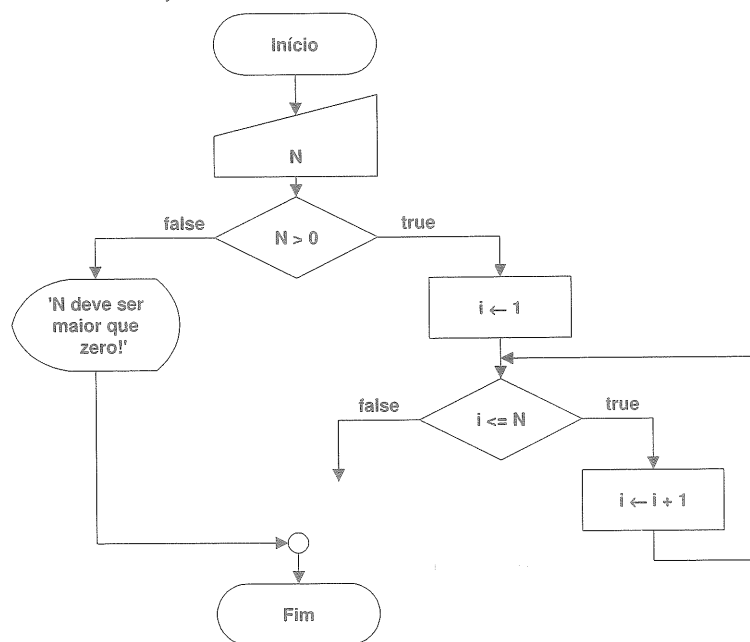



Figura 3.20 Passo 2 na construção do fluxograma para o problema das temperaturas.

Deve-se observar que a instrução $i \leftarrow i+1$ é interpretada da seguinte forma: soma-se 1 ao valor da variável i e então atribui-se esse valor à própria variável i. Como uma variável representa uma área da memória que contém um valor, nada impede que esse valor seja somado a 1 e então gravado de volta na mesma posição.

Agora que já existe uma forma de repetir N vezes alguma instrução, este é o momento de introduzir o comando de leitura da temperatura. Utilizando a variável T para representar uma temperatura, coloca-se o comando de leitura no interior da repetição apresentada, de acordo com a Figura 3.21.

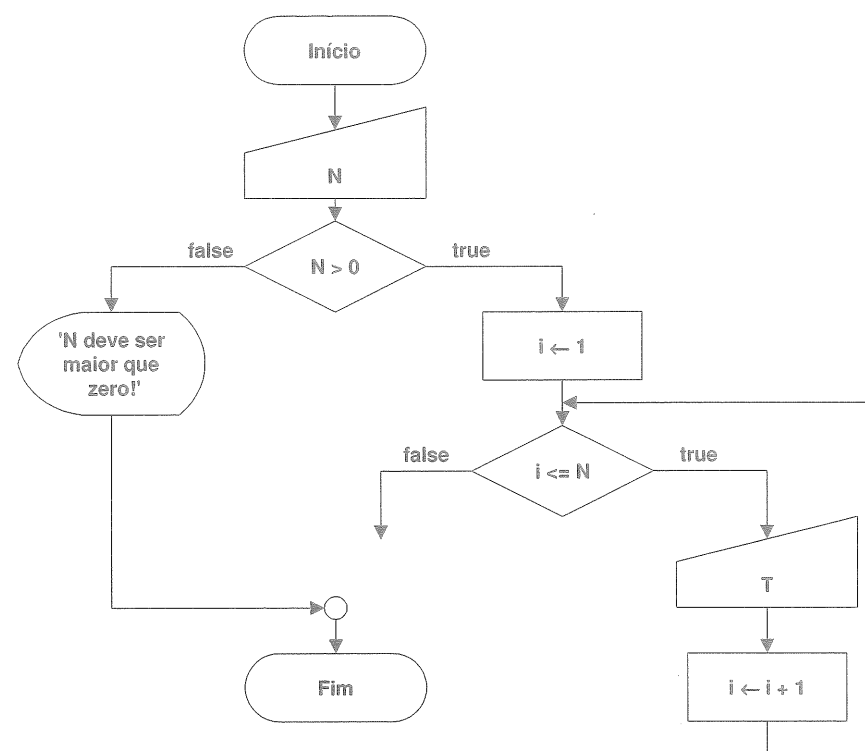



Figura 3.21 Passo 3 na construção do fluxograma para o problema das temperaturas.

O fluxograma da Figura 3.21 permite a entrada de N temperaturas, porém, armazena-se apenas o último valor delas, pois se está utilizando uma única variável para guardar a temperatura, a variável T. No próximo passo da construção do fluxograma, deve-se adicionar um comando que possibilite a soma dessa variável com a soma parcial existente anteriormente.

Para se fazer isso, basta acrescentar mais uma variável ao problema, a qual armazenará a soma das temperaturas (uma variável com esse propósito é denominada

acumulador). A técnica para isso é iniciar uma variável fora do comando de repetição com o valor zero, por exemplo, $S \leftarrow 0$ e, então, após a leitura de uma temperatura, somar essa temperatura a essa variável, como em $S \leftarrow S + T$. Assim, a cada repetição, lê-se uma temperatura e, então, prontamente é feita sua soma com a soma anterior.

Por fim, após a execução das N somas, basta dividir o valor da variável S por N, obtendo-se a média e, então, exibindo esse valor para o usuário. O fluxograma final está representado na Figura 3.22.

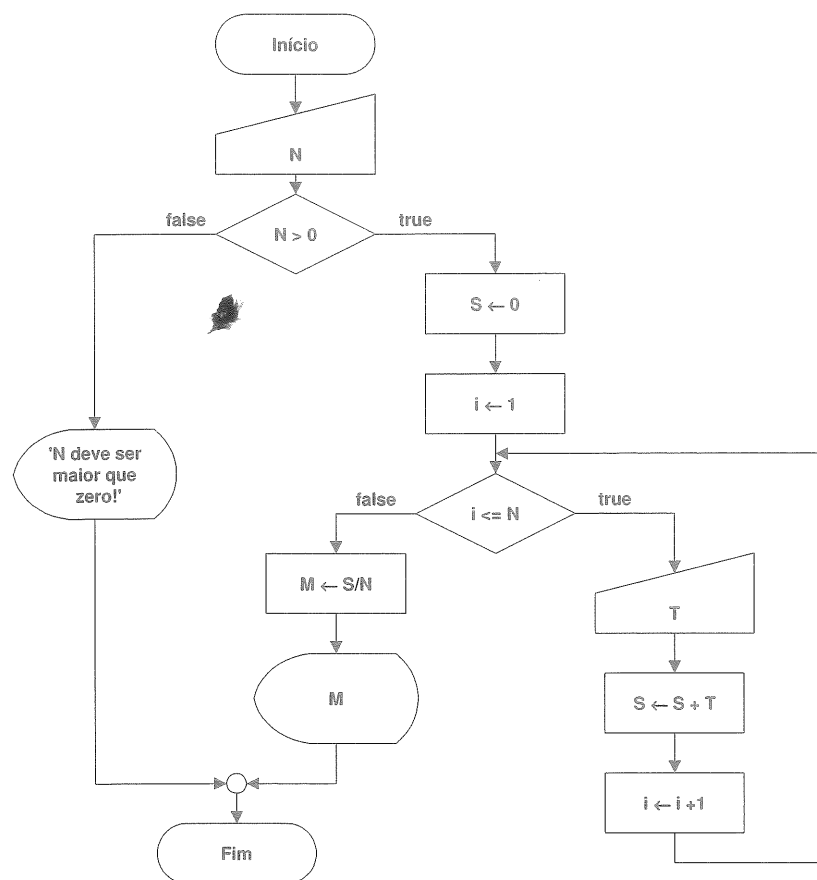



Figura 3.22 Fluxograma final para o problema das temperaturas.

Agora é o momento de verificar se esse fluxograma está correto ou não. É necessário realizar testes com diversos valores de entrada (corretos ou não) e, ao simular o fluxograma, sempre alcançar o símbolo **Fim** com valores coerentes.

Serão definidos dois conjuntos de testes:

- com valores de entrada suspeitos: $N = -1$ e temperaturas da lista $(10, 11, 12)$;
- com valores de entrada corretos: $N = 3$ e temperaturas da lista $(10, 11, 12)$.

Utilizando o primeiro conjunto de valores e seguindo o Algoritmo 3.3 de interpretação de fluxogramas, obtém-se a Tabela 3.3.

Tabela 3.3 Simulação do fluxograma com valores de entrada errados.

		Valor das variáveis				
Passo	**Comando**	N	S	i	T	M
1	Entrada (digitar: -1)	-1	?	?	?	?
2	Decisão (*false*: $N < 0$)	-1	?	?	?	?
3	Exibição (exibir: 'N deve ser maior que zero!')	-1	?	?	?	?

Nesse caso, o fluxograma apresentou como resposta a exibição da mensagem 'N deve ser maior que zero!' e terminou. Isso era esperado, já que para os valores negativos de N não deveria ser produzida nenhuma média. Realizando a simulação com o segundo conjunto de valores de entrada, obtém-se a Tabela 3.4.

Observe que por essa simulação verificou-se que o fluxograma da Figura 3.22 fornece os resultados corretos também. Basta checar: com $N = 3$ valores digitados que foram $(10, 11, 12)$ foi produzido e exibido o valor $M = 11$, que é a média desses valores particulares. É interessante, para praticar, testar esse mesmo fluxograma com outros valores de teste.

3.6 Convenções para tipos de dados

Um fluxograma não indica explicitamente o tipo do valor que opera. Números como 4 e 67 podem ser utilizados como números inteiros ou reais. Por sua vez, números como 45,78 e 55,0987 são com certeza números reais. Existem ainda os tipos para as cadeias de caracteres, caracteres simples e valores lógicos. Assim, com o objetivo de implementar futuramente os algoritmos em uma linguagem de programação, será convencionado o formato desses tipos de dados.

Tabela 3.4 Simulação do fluxograma com valores de entrada corretos.

		Valor das variáveis				
Passo	**Comando**	N	S	i	T	M
1	Entrada (digitar: 3)	3	?	?	?	?
2	Decisão (true: $N > 0$)	3	?	?	?	?
3	Processo ($S \leftarrow 0$)	3	0	?	?	?
4	Processo ($i \leftarrow 1$)	3	0	1	?	?
5	Decisão (true: $i <= N$)	3	0	1	?	?
6	Entrada (digitar: 10)	3	0	1	10	?
7	Processo ($S \leftarrow S + T$)	3	10	1	10	?
8	Processo ($i \leftarrow i + 1$)	3	10	2	10	?
9	Decisão (true: $i <= N$)	3	10	2	10	?
10	Entrada (digitar: 11)	3	10	2	11	?
11	Processo ($S \leftarrow S + T$)	3	21	2	11	?
12	Processo ($i \leftarrow i + 1$)	3	21	3	11	?
13	Decisão (true: $i <= N$)	3	21	3	11	?
14	Entrada (digitar: 12)	3	21	3	12	?
15	Processo ($S \leftarrow S + T$)	3	33	3	12	?
16	Processo ($i \leftarrow i + 1$)	3	33	4	12	?
17	Decisão (false: $i > N$)	3	33	4	12	?
18	Processo ($M \leftarrow S/N$)	3	33	4	12	11
19	Exibição (exibir: 11)	3	33	4	12	11

3.6.1 Números

Os números manipulados em um algoritmo podem ser inteiros ou reais. Os números inteiros são escritos sem separador de decimal, como, por exemplo, 10, 334, 13 etc. Os números reais serão escritos separando-se a parte decimal com um *ponto*. Portanto, entende-se como números reais os valores como 3.1415, 45.98, 1.0 etc.

Alternativamente, esses números reais (principalmente quando muito grandes) podem ser escritos em notação científica (ou notação exponencial), expressando o número em potências de dez. Dessa forma, a constante de Avogrado, 6.02×10^{23}, pode ser escrita em um algoritmo com a seguinte notação científica: $6.02E23$. Utiliza-se, por conseguinte, o símbolo E para separar o número de sua potência de dez.

Para os números negativos, precede-se o número em questão com o sinal "$-$". Assim, são exemplos de números negativos: -3, -0.9, -12 etc. Em notação científica, se o expoente for negativo, precede-se seu valor com o sinal "$-$". Exemplos: $4.5E-3$, $-3.5E-45$ etc.

3.6.2 Caracteres e cadeias de caracteres

Os caracteres representam uma única letra ou símbolo de texto. Os caracteres em um algoritmo serão representados limitando-os com apóstrofes. Será utilizada essa convenção para evitar que haja confusões entre os caracteres e os nomes de variáveis. Dessa maneira, o caractere W será representado em um algoritmo como 'W'. Com essa notação, não há perigo de confundir h, que se entende por nome de variável, com 'h', que se entende pelo caractere h minúsculo.

Os símbolos fora do alfabeto também são representados como caracteres: '@', '\', '/' etc. O espaço em branco entra nessa categoria, também: ' '. As cadeias de caracteres ou simplesmente *strings* (cordões de caracteres) representam um conjunto ordenado de caracteres, significando palavras, mensagens ou ainda pequenos textos. Uma cadeia de caracteres será representada, limitando-a com apóstrofes. Assim, são exemplos de cadeias de caracteres: 'Olá', '123XYZ', '098-33#' etc.

As cadeias de caracteres podem conter quaisquer caracteres, incluindo os espaços em branco. Dessa forma, as cadeias 'Bom dia', 'Erro no Aplicativo' contêm espaços em branco. Deve-se ter cuidado com a diferença entre os números e uma cadeia de caracteres que contém caracteres que representam números. São completamente diferentes os valores 123, que é um inteiro, de '123', que é uma cadeia que contém os caracteres '1', '2', e '3'.

Normalmente, em linguagens de programação, existe uma limitação para o número de caracteres em uma cadeia. Será convencionado o limite de 255 caracteres.

3.6.3 Valores lógicos

Os valores lógicos ou ainda booleanos (em homenagem a George Boole, que elaborou a lógica booleana) são aqueles que representam apenas dois estados: um estado verdadeiro ou um estado falso. Os valores booleanos serão convencionados segundo a grafia inglesa. Assim, o valor **verdadeiro** será escrito ***true*** e o valor **falso**, ***false***. O uso desses valores em expressões lógicas será estudado adiante.

3.7 Convenções para os nomes de variáveis

As variáveis utilizadas em um algoritmo devem ser escritas de modo claro, inteligível. Nesse sentido, convencionam-se algumas regras para nomear as variáveis:

1. Os nomes podem ter até 63 caracteres de comprimento.

2. Os nomes devem ser iniciados por um caractere alfabético (letra) ou pelo caractere '_'.
3. Os nomes podem possuir números, desde que se inicie por letra.
4. Além de letras, números e o caractere '_', não é aceito nenhum outro símbolo.
5. Letras fora do alfabeto ocidental, como as letras gregas, não são aceitas.
6. Não se fará diferenciação entre as letras maiúsculas e minúsculas nos nomes de variáveis.

Exemplos de nomes válidos de variáveis: A, $Ba12$, $A1543T$, $CustoTotal$, $Pagamento$, $Nome$, $Lista$, $Contador$, $Valores$, $Indice$, $Resultado_Final$, $C3PO$, $NCC1701$, $R2D2$.

Exemplos de nomes inválidos: $1QA$, por começar com valor numérico; $Preço$, por possuir caracter especial (ç); $C\&A$ pelo mesmo motivo anterior (&); $Resultado\ Final$ (espaço em branco); $Media-final$, pois o sinal de menos é reservado para a operação subtração.

3.8 Convenções para as expressões

3.8.1 Operação de atribuição

A atribuição é a operação que permite armazenar um valor em uma variável. Para essa operação, será utilizado o símbolo ← (seta para a esquerda), e a variável que receberá esse valor ou resultado de uma expressão deverá estar do lado esquerdo da seta.

Dessa forma, pode-se ler a expressão $A \leftarrow 6$ como **armazenar o valor 6 na variável** A ou ainda como **A recebe 6** . Ainda é possível se escrever $A \leftarrow A + 1$ que significa A **recebe o valor de** $A + 1$. O símbolo ← foi escolhido para evitar confusões com o operador =, reservado para a realização de comparações.

Em um fluxograma, a atribuição somente pode ser utilizada em blocos que representam processos ou comandos (veja a Figura 3.23).

Nesse exemplo, ao se entrar com um valor para a variável A, será obtido como resultado o valor original acrescido de 1. Se A for 5, o resultado exibido será 6.

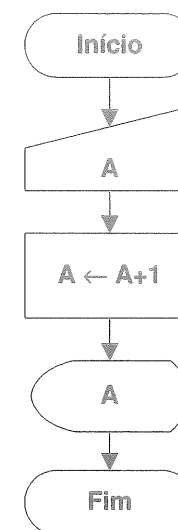



Figura 3.23 Exemplo do comando de atribuição.

3.8.2 Operações aritméticas

São definidas as quatro operações aritméticas: adição, subtração, multiplicação e divisão. Dependendo dos operadores e do tipo de resultado, a divisão pode ter de ser feita de forma diferente: a **divisão real**, a **divisão inteira** e o **resto da divisão inteira**. Essas operações matemáticas são consideradas operações binárias, pois envolvem sempre dois operandos.

A operação troca de sinal é unária, pois altera o sinal de um único operando. Um resumo das operações aritméticas é apresentado na Tabela 3.5.

As operações aritméticas apresentadas abrangem os números inteiros e os números reais. No entanto, será convencionado que a operação de adição (+) também poderá ser aplicada à cadeia de caracteres. Nesse caso, representa a operação de **concatenação**, isto é, permite a junção de duas ou mais cadeias de caracteres. Por exemplo, se a variável A contém a cadeia 'República ' (existe um espaço após a última letra) e a variável B contém a cadeia 'Federativa', a operação $C \leftarrow A{+}B$ resultará em C a cadeia 'República Federativa'.

Como exemplo do uso de operações aritméticas com números, segue na Figura 3.24 o fluxograma que, dado dois números inteiros A e B, calcula o cociente Q e o resto R da divisão inteira de A por B.

Tabela 3.5 Operações aritméticas.

Operação	1º operando (A)	2º operando (B)	Tipo resultante (C)	Simbologia
Adição	Inteiro	Inteiro	Inteiro	$C \leftarrow A + B$
	Real	Real	Real	
	Real	Inteiro	Real	
	Inteiro	Real	Real	
Subtração	Inteiro	Inteiro	Inteiro	$C \leftarrow A - B$
	Real	Real	Real	
	Real	Inteiro	Real	
	Inteiro	Real	Real	
Multiplicação	Inteiro	Inteiro	Inteiro	$C \leftarrow A * B$
	Real	Real	Real	
	Real	Inteiro	Real	
	Inteiro	Real	Real	
Divisão real	Inteiro	Inteiro	Real	$C \leftarrow A/B$
	Real	Real	Real	
	R[illegible]	Inteiro	Real	
	Inteiro	Real	Real	
Divisão inteira	Inteiro	Inteiro	Inteiro	$C \leftarrow A\ div\ B$
Resto	Inteiro	Inteiro	Inteiro	$C \leftarrow A\ mod\ B$
Troca de sinal	Inteiro	Não aplicável	Inteiro	$C \leftarrow -A$
	Real	Não aplicável	Real	

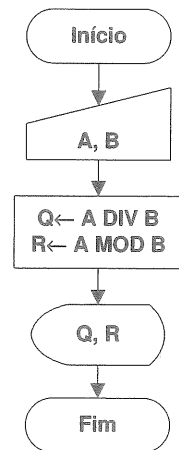



Figura 3.24 Exemplo de operadores aritméticos.

3.8.3 Operações relacionais

As operações relacionais permitem efetuar comparações entre duas variáveis. Essas operações são largamente utilizadas em estruturas condicionais e em repetitivas. Já que essas operações dependem de testes de valores e, **todo resultado de um teste só pode ser *true* ou *false***, seu resultado é um valor **booleano**. Os operadores relacionais são apresentados na Tabela 3.6.

Tabela 3.6 Operações relacionais.

Operador	Significado	Resultado
=	Igualdade	Booleano (*true* ou *false*). Sendo *true*, se o 1º operando for igual ao 2º e *false*, se diferentes.
<	Menor	Booleano (*true* ou *false*). Sendo *true*, se o 1º operando for menor que o 2º e *false*, caso contrário.
>	Maior	Booleano (*true* ou *false*). Sendo *true*, se o 1º operando for maior que o 2º e *false*, caso contrário.
<=	Menor ou igual	Booleano (*true* ou *false*). Sendo *true*, se o 1º operando for menor ou igual ao 2º e *false*, caso contrário.
>=	Maior ou igual	Booleano (*true* ou *false*). Sendo *true*, se o 1º operando for maior ou igual ao 2º e *false*, caso contrário.
<>	Diferente	Booleano (*true* ou *false*). Sendo *true*, se o 1º operando for diferente do 2º e *false*, caso sejam iguais.

As operações relacionais também são definidas para as cadeias de caracteres. Nesse caso, vale a avaliação **lexicográfica**, como nos dicionários (mas se baseando na ordem da tabela ASCII). Assim, se a variável A possui a cadeia ‘Banana’ e se a variável B contém ‘Banana d’água’, então $A < B$ é $true$; $A > B$ é $false$ e assim por diante. A comparação de cadeias é sensível ao caso: ‘Banana’<>‘banana’ e ‘Banana’<‘banana’ (pois o caractere ‘B’ tem código ASCII menor que o caractere ‘b’).

3.8.4 Operações lógicas

São operações efetuadas com os valores booleanos (***true*** ou ***false***), resultando em valores booleanos. Essas operações são largamente utilizadas em estruturas condicionais e em repetitivas, já que essas estruturas dependem de testes de valores. Os operadores lógicos têm a função de advérbio na linguagem de programação. Cada operador lógico tem a

sua "tabela verdade" composta por: ***not***, ***and*** e ***or***.

O operador ***not*** representa a negação do valor booleano – sua inversão. O operador ***and*** realiza o teste simultâneo de duas condições – resulta o valor $true$ somente quando ambos operandos são $true$. Já para o operador ***or***, basta que uma das condições seja $true$, para que o resultado seja $true$. A Tabela 3.7 exibe a tabela verdade dos operadores lógicos.

Tabela 3.7 Operações lógicas.

Operador lógico	1º operando (A)	2º operando (B)	Resultado (C)	Simbologia
NOT	*true*	Não aplicável	*false*	$C \leftarrow not\ A$
	false	Não aplicável	*true*	
AND	*true*	*true*	*true*	$C \leftarrow A\ and\ B$
	true	*false*	*false*	
	false	*true*	*false*	
	false	*false*	*false*	
OR	*true*	*true*	*true*	$C \leftarrow A\ or\ B$
	true	*false*	*true*	
	false	*true*	*true*	
	false	*false*	*false*	

Assim, considerando-se as seguintes variáveis com valores iniciais indicados, $A = 3$, $B = 7$, $C = -1$, a expressão $not((C > 0)\ and\ ((B - A) > 2))$ resulta em $true$, pois $C > 0$ é $false$ e $((B - A) > 2)$ é $true$, resultando a operação *and* aplicado nessas expressões em $false$. Por fim, o resultado $not(false)$ fornece seu inverso lógico, ou seja, $true$.

3.8.5 Expressões

Se prestarmos atenção, todas as operações discutidas são levadas a efeito com um ou dois operandos de cada vez. Uma expressão contendo diversos operandos deve ser avaliada de acordo com a **precedência** dos operadores envolvidos. A precedência dos operadores indica, em uma expressão, qual operação será realizada antes das outras.

Os operadores aritméticos possuem a seguinte precedência, do maior para o menor:

1. Troca de sinal ("$-$" unário).
2. $*$, $/$, mod, div na ordem em que aparecerem.
3. $+$ e $-$.

Por exemplo, a expressão $A \leftarrow 24/6/2 - 127\ div\ 7\ mod\ 5$ deve se assim avaliada:

$$A \leftarrow 2 - 18\ mod\ 5$$
$$A \leftarrow 2 - 3$$
$$A \leftarrow -1$$

De forma diferente da Matemática, na qual as expressões podem ser agrupadas com chaves, colchetes e parênteses, aqui o único elemento de agrupamento de expressões válido serão os parênteses. Assim, em expressões contendo parênteses, a precedência é ligeiramente alterada:

1. Resolver o nível mais interno de parênteses, usando a precedência de operações:
   (a) Troca de sinal ("$-$" unário).
   (b) $*$, $/$, mod, div na ordem em que aparecerem.
   (c) $+$ e $-$.
2. Passar para o próximo nível, sempre do mais interno para o mais externo, utilizando a precedência de operações.

No caso de operadores lógicos, a precedência é (do maior para o menor): ***not***, ***and*** e por fim ***or***. Esta última também pode ser alterada, da mesma forma, pelo uso de parênteses. Finalmente, em um último nível da hierarquia dos operadores, têm-se os operadores relacionais.

A Tabela 3.8 fornece a precedência de todos os operados vistos até este ponto.

Tabela 3.8 Precedência dos operadores.

Hierarquia	Operadores
1	not
2	$*$, $/$, div, mod, and
3	$+$, $-$, or
4	$>$, $>=$, $<$, $<=$, $=$, $<>$

3.9 Sub-rotinas predefinidas

Até este ponto foram apresentados os tipos de dados básicos para a construção de fluxogramas e suas operações. No entanto, surge um problema, caso seja necessário definir as

operações com funções matemáticas conhecidas, como *seno*, *logaritmos* e outras. Além disso, a única operação que foi apresentada para as cadeias de caracteres foi a concatenação. Como fazer, então, para se extrair parte de um nome, por exemplo?

Em princípio, todas as operações necessárias poderiam ser definidas via novos algoritmos. Entretanto, este é um trabalho desnecessário, pois a maior parte das linguagens de programação (em uma continuação do estudo apresentado neste livro) mostra um conjunto de **sub-rotinas** que resolvem problemas básicos matemáticos e de cadeias de caracteres.

Considere, por enquanto, como uma sub-rotina um algoritmo que foi escrito e que está pronto para o uso. Uma sub-rotina pode apresentar **parâmetros**, que servem para se passar valores a serem calculados. Uma sub-rotina tipo **função** é aquela que produz um valor diretamente. E uma sub-rotina tipo **procedimento** é a que não produz um valor ou produz um valor indiretamente, via um de seus parâmetros. Os conceitos de como escrever novas sub-rotinas é assunto do Capítulo 6.

Nas seções a seguir serão convencionadas as sub-rotinas válidas para serem utilizadas neste livro. Qualquer operação não mencionada nas tabelas a seguir poderão ser implementadas com o uso de uma ou mais sub-rotinas apresentadas.

3.9.1 Funções matemáticas

Consideram-se válidas as seguintes funções predefinidas, comuns a várias linguagens de programação de alto nível, expostas na Tabela 3.9.

Os argumentos dessas funções podem ser números reais ou inteiros e o tipo de resultado produzido é apresentado na própria tabela. Dessa tabela, seguem algumas observações. Na primeira delas, as funções int e $frac$ servem para "destrinchar" números reais, resultando em novos números reais. Por exemplo, a expressão $A \leftarrow int(3.1415)$ resulta em $A = 3.0$ e a expressão $A \leftarrow frac(3.1415)$, em $A = 0.1415$. Já as funções $trunc$ e $round$ servem, respectivamente, para truncar e arredondar um número real em um número inteiro. Dessa forma, a expressão $A \leftarrow trunc(3.1415)$ resulta em $A = 3$ (inteiro), e a expressão $A \leftarrow round(3.1415)$, em $A = 3$ (o arredondamento é realizado para o inteiro mais próximo).

Na segunda, as funções trigonométricas apresentadas utilizam e resultam ângulos em radianos. Assim, caso seja necessário calcular o seno de 33°, esse arco deverá ser convertido em radianos com a expressão $33 \times \pi/180$, resultado em ≈ 0.58 radianos.

Um exemplo de aplicação das funções dessa tabela é o fluxograma para calcular x^y para todo $x > 0$. Nota-se que não existe uma função pronta para realizar esse cálculo. No entanto, uma expressão equivalente utilizando funções da Tabela 3.9 pode ser desenvolvida, considerando-se as seguintes igualdades:

Tabela 3.9 Funções matemáticas.

Função	**Utilidade**	**Tipo do resultado**
$ln(x)$	Retorna o valor do logaritmo neperiano de x.	Real
$exp(x)$	Retorna o valor de e^x.	Real
$int(x)$	Retorna a parte inteira de x.	Real
$frac(x)$	Retorna a parte fracionária de x.	Real
$trunc(x)$	Retorna a parte inteira de x.	Inteiro
$round(x)$	Arredonda x para o próximo inteiro.	Inteiro
$sqr(x)$	Retorna x^2.	Real
$sqrt(x)$	Retorna $\sqrt{x}$.	Real
$sin(x)$	Retorna o seno de x.	Real
$cos(x)$	Retorna o cosseno de x.	Real
$arctan(x)$	Retorna o arco, em radianos, cuja tangente é x.	Real
$abs(x)$	Retorna o módulo de x ($\|x\|$).	Real ou Inteiro

$$
\begin{aligned}
z &= x^y \\
\ln(z) &= y \ln x \\
z &= e^{y \ln x}
\end{aligned}
$$

Assim, a expressão para calcular x^y, utilizando as funções da Tabela 3.9, é $exp(y * ln(x))$. Entretanto, deve-se prestar atenção para valores negativos na base, ou seja, para a variável x. Um fluxograma bem simples para calcular o valor de x e y fornecidos está descrito na Figura 3.25.

3.9.2 Funções e procedimentos para as cadeias de caracteres

As cadeias de caracteres também podem ser manipuladas. Convencionam-se as operações de cadeias de caracteres segundo a Tabela 3.10.

Por exemplo, um algoritmo que lê o nome de um estudante e então exibe a mensagem de que ele será aprovado em computação, ficaria assim representado pela Figura 3.26.

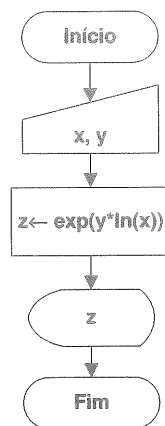



Figura 3.25 Exemplo do uso de funções matemáticas.

Tabela 3.10 Funções e procedimentos para as cadeias de caracteres.

Operação	Significado	Exemplos
$length(s)$	Fornece como resultado o número de caracteres que compõe uma cadeia S.	$N \leftarrow length($'Olá'$)$ O valor de N é 3.
$concat(S1, S2)$	Une duas cadeias, a 2ª ($S2$) no final da 1ª ($S1$), formando uma nova cadeia.	$S \leftarrow concat($'Bom','Dia'$)$ O valor de S é 'BomDia'.
$copy(S, ini, num)$	Retorna (copia) a uma nova cadeia de caracteres com os num elementos da cadeia S a partir da posição ini.	$S \leftarrow copy($'Turbo Pascal'$, 7, 6)$ O valor de S é 'Pascal'.
$insert(S1, S2, ini)$	Insere uma nova cadeia ($S1$) na posição ini de $S2$, deslocando para a esquerda o resto da cadeia original ($S2$).	$S \leftarrow$'Turbo 7.0' $insert($'Pascal'$, S, 7)$ O valor de S é 'Turbo Pascal7.0'
$pos(S1, S2)$	Fornece como resultado a posição, na qual a cadeia $S1$ começa dentro da cadeia $S2$. Se a cadeia $S1$ não existir em $S2$, o resultado será zero.	$I \leftarrow pos($'Pascal','Turbo Pascal'$)$ Aqui o valor de I é 7, mas $I \leftarrow pos($'pascal','Turbo Pascal'$)$ O valor de I é 0. Existe diferença entre maiúsculas e minúsculas nessa função.

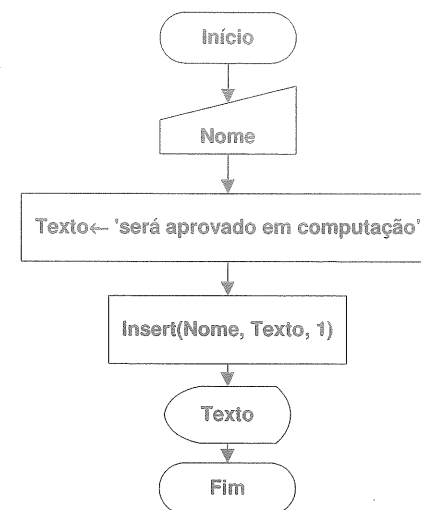



Figura 3.26 Exemplo do uso de operações com cadeias de caracteres.

Se o valor da variável $Nome$ for 'Douglas', o valor da variável $Texto$ a ser apresentado será 'Douglas será aprovado em Computação'.

3.10 Exercícios

3.1. Elabore um fluxograma que calcule quantas notas de 50, 10 e 1 são necessárias para se pagar uma conta cujo valor é fornecido.

3.2. Reescreva as expressões abaixo, utilizando as convenções adotadas para os fluxogramas:

a) $C = 3 \cdot \dfrac{(5+3)}{2 \cdot 3} - 4 + 7$

b) $q = \sqrt{\dfrac{g \cdot P_m^5}{L \cdot V_m}}$

c) $\eta = 1 - \dfrac{\omega}{\mu} \cdot \dfrac{tg\ \epsilon}{sen(\beta + \epsilon)}$

d) $k_0 = \dfrac{3 \cdot 10^{-7}}{\sqrt[5]{Re}}$

e) $D = \frac{3}{2 + \frac{5}{2+\frac{1}{3}}}$

f) $L_p = \frac{(d-t)^4 \cdot \pi}{32}$

g) $H_{vl} = T_A \cdot \frac{\omega^2}{2 \cdot g} \cdot \frac{L}{t} \cdot \frac{tg\,\varepsilon}{sen\,\beta}$

3.3. ☼ Resolva as expressões abaixo, destacando o resultado final:

a) $A \leftarrow (18/3/2 - 1) * 5 - 4 - (2 + 3 + 5)/2$

b) $B \leftarrow 26/6/2 - 127\ div\ 7\ mod\ 5$

c) $C \leftarrow 7\ mod\ 4 - 8/(3 + 1)$

d) $D \leftarrow (2 >= 5)\ and\ (1 <> 0)\ and\ not(6 <= 2 * 3)\ or\ (10 <> 10)$

e) $E \leftarrow (5 <> 2)\ or\ not(7 > 4)\ and\ (4 <= pi)$

3.4. ☼ Considerando as seguintes atribuições, $R \leftarrow 2$, $S \leftarrow 5$, $T \leftarrow -1$, $X \leftarrow 3$, $Y \leftarrow 1$ e $Z \leftarrow 0$, resolver as expressões abaixo:

a) $A \leftarrow (R >= 5)\ or\ (T > Z)\ and (X - Y + R > 3 * Z)$

b) $B \leftarrow (abs(T) + 3 >= 4)\ and\ not(3 * R/2 < S * 3)$

c) $C \leftarrow (X = 2)\ or\ (Y = 1)\ and\ ((Z = 0)\ or\ (R > 3))\ and\ (S < 10)$

d) $D \leftarrow (R <> S)\ or\ not(sqrt(R) < sqrt(X))\ and\ (4327 * X * S * Z = 0)$

3.5. ☼ ✎ Elabore um fluxograma que calcule o alcance de um projétil, dada a velocidade inicial v_0 e o ângulo θ entre o cano do canhão e o solo. A fórmula a ser utilizada é:

$$S = \frac{{v_0}^2}{g} sen(2\theta)$$

3.6. ☼ Elabore um fluxograma que calcule a área de um triângulo pela fórmula de Hierão:

$$K = \sqrt{s(s-a)(s-b)(s-c)}$$

em que K é a área do triângulo, s o semiperímetro e a, b e c são os lados do triângulo.

3.7. ☼ Elabore um fluxograma que permita a entrada de um número inteiro e diga se ele é par ou ímpar.

3.8. ☼ Elabore um fluxograma que leia dois números (x e y) e escreva como resultado o maior entre eles.

3.9. ☼ ✎ Elabore um fluxograma que permita a entrada de dois valores, x e y, troque seus valores entre si e então exiba os novos resultados.

3.10. ☼ O mesmo do Exercício 3.9, com as seguintes exigências: só podem ser utilizadas duas variáveis e operações de adição e subtração.

3.11. ☼ ✎ Elabore um fluxograma que permita a entrada de n (lido pelo teclado) valores reais e apresente como resultado o maior entre esses valores.

3.12. ☼ Idem ao Exercício 3.8, só que escreva o menor deles.

3.13. ☼ Elabore um fluxograma que calcule e exiba a média de dois números digitados.

3.14. ☼ ✎ Elabore um fluxograma que calcule e exiba a média de 500 números fornecidos pelo usuário.

3.15. ☼ Elabore um fluxograma que calcule e exiba a soma dos números pares contidos entre zero e um número par fornecido via teclado.

3.16. ☼ Elabore um fluxograma que calcule e exiba a soma dos números ímpares contidos entre zero e um número ímpar fornecido via teclado.

3.17. ☼ A contribuição para o INSS (interessante para estrutura condicional por ser progressivo) é calculada a partir da tabela a seguir :

TABELA VIGENTE **Tabela de contribuição dos segurados empregado, empregado doméstico e trabalhador avulso, para pagamento de remuneração a partir de 16 de junho de 2010** **Portaria nº 408, de 17 de agosto de 2010**	
Salário de contribuição (R$)	**Alíquota para fins de recolhimento ao INSS (%)**
até R$ 1.040,22	8,00 %
de R$ 1.040,23 a R$ 1.733,70	9,00 %
de R$ 1.733,71 até R$ 3.467,40	11,00 %
acima de R$ 3.467,40	valor fixo de R$ 381,41

Elabore um algoritmo que, para uma entrada do salário bruto, calcule a contribuição ao INSS e o salário líquido restante.

3.18. O desconto do IRRF (Imposto de Renda Retido na Fonte), também denominado "Mordida do Leão", é calculado sobre o salário líquido após a dedução da contribuição ao INSS, de acordo com a seguinte tabela:

Base de Cálculo em R$	Alíquota %	Parcela a Deduzir do Imposto em R$
Até 1.499,15	–	–
De 1.499,16 até 2.246,75	7,5	112,43
De 2.246,76 até 2.995,70	15	280,94
De 2.995,71 até 3.743,19	22,5	505,62
Acima de 3.743,19	27,5	692,78

Líquido = Bruto-INSS-IR
Líquido = Bruto-INSS-(base*alíquota - parcela)

Elabore um fluxograma que, para uma entrada do salário bruto e após a dedução da contribuição (veja o Exercício 3.17), calcule o desconto do IRRF.

3.19. Elabore um fluxograma que leia as quatro notas de prova ($P1$, $P2$, $P3$ e $P4$) e quatro notas de trabalho ($T1$, $T2$, $T3$ e $T4$) e posteriormente exiba a mensagem 'Aprovado' ou 'Não aprovado' dependendo dos valores obtidos, conforme as regras de cálculo definidas a seguir:

- Média de provas: $MP = \frac{P1 + P2 + P3 + P4}{4}$
- Média de trabalhos: $MT = \frac{T1 + T2 + T3 + T4}{4}$
- Média final: $MF = 0{,}8MP + 0{,}2MT$
- Situação:
  - se $MF \geq 6{,}0 \Rightarrow$ aprovado;
  - se $MF < 6{,}0 \Rightarrow$ não aprovado.

3.20. Elabore um fluxograma que transforme uma temperatura fornecida em °C para a correspondente em °F. A fórmula de conversão de °F para °C é:

$$°C = \frac{5}{9}(°F - 32)$$

3.21. Elabore um fluxograma que permita a entrada de dois valores inteiros e faça uma contagem decrescente desde o maior deles até o menor.

3.22. Elabore um fluxograma que permita a entrada de três valores e faça a contagem desde o primeiro deles até o segundo com passo dado pelo terceiro.

3.23. Um número inteiro é considerado triangular se este for o produto de três números inteiros consecutivos, como, por exemplo $120 = 4 \times 5 \times 6$. Elabore um fluxograma que, após ler um número n, verifique se o mesmo é ou não triangular.

3.24. Elabore um fluxograma que leia um valor n inteiro e verifique se este é ou não primo (número primo é divisível por um e por ele mesmo).

3.25. Um número palíndromo é aquele que se lido da esquerda para a direita e da direita para a esquerda possui o mesmo valor (por exemplo: 34543). Elabore um fluxograma que leia um número n, inteiro, e verifique se ele é um palíndromo.

3.26. Elabore um fluxograma que receba três valores digitados A, B e C, informando se estes podem ser os lados de um triângulo. O ABC é triângulo se $A < B + C$ e $B < A + C$ e $C < A + B$.

3.27. Elabore um fluxograma que permita a entrada de 30 valores e mostre a soma de seus inversos. Observação: o inverso de x é $1/x$.

3.28. Elabore um fluxograma que permita a entrada de n valores e mostre a soma de seus quadrados.

3.29. Elabore um fluxograma que permita a entrada de dez valores e calcule o produto de todos eles.

3.30. Elabore um fluxograma que represente o algoritmo para calcular a soma dos primeiros 40 termos da sequência definida a seguir, com o valor de A fornecido via teclado:

$$\frac{7 \cdot A}{3}, \frac{7 \cdot A}{6}, \frac{7 \cdot A}{12}, \frac{7 \cdot A}{24}, \frac{7 \cdot A}{48}, \ldots$$

3.31. Elabore um fluxograma que represente o algoritmo para calcular a soma dos primeiros N termos da sequência definida a seguir, com N fornecido via teclado:

$$\frac{1}{2}, \frac{1}{4}, \frac{1}{6}, \frac{1}{8}, \frac{1}{10}, \ldots$$

3.32. Elabore um fluxograma que represente o algoritmo para calcular a soma dos primeiros N termos da sequência definida a seguir, com N fornecido via teclado:

$$S = 1 + 2 + 3 + 4 + 5 + \ldots + N$$

3.33. O número π pode ser calculado por meio da série infinita:

$$\pi = 4 \cdot (1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \frac{1}{11} + \frac{1}{13} - \ldots)$$

Elabore um fluxograma que calcule e exiba o valor do número π, utilizando a série anterior, até que o valor absoluto da diferença entre o número calculado em uma iteração e o da anterior seja menor ou igual a 0.00000000005.

3.34. Elabore um fluxograma que, dados dois números complexos $c1$ e $c2$, calcule as seguintes operações: soma, subtração e multiplicação.

Lembrando: um número complexo possui duas partes, uma real (re) e uma imaginária (im), representado genericamente como $c = re + j \cdot im$.

3.35. O número 3025 possui uma característica interessante, sendo a seguinte: $30 + 25 = 55$ e $55^2 = 3025$. Elaborar um fluxograma que verifique se um número inteiro de quatro algarismos (digitado) tem essa propriedade ou não.

3.36. Elabore um fluxograma que represente o algoritmo do cálculo dos 50 primeiros termos da série apresentada a seguir, sendo X um valor digitado:

$$\frac{2 \cdot 3}{X+3}, \frac{2 \cdot 5}{X+5}, \frac{2 \cdot 7}{X+7}, \frac{2 \cdot 9}{X+9}, \ldots$$

3.37. Qual o resultado exibido pelo fluxograma da Figura 3.27, se o dado de entrada digitado for sua idade?

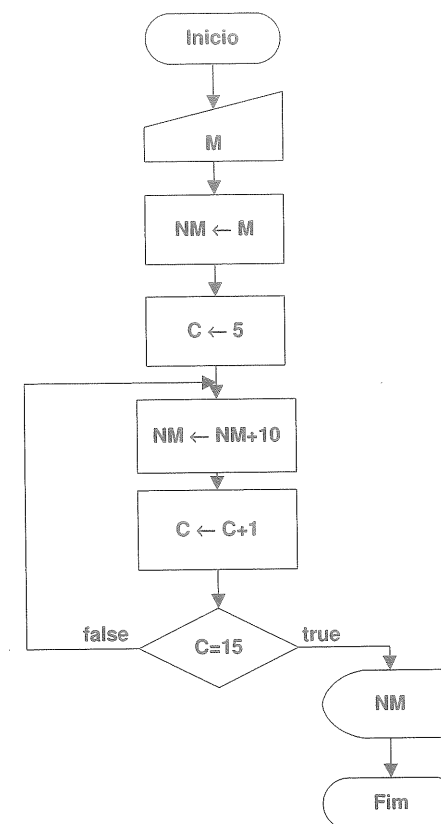



Figura 3.27 Fluxograma para o Exercício 3.37.

3.38. Elabore um fluxograma que calcule e exiba a tensão S de uma barra cilíndrica de diâmetro D submetida a uma carga Q. Os valores de D e Q devem ser digitados via teclado. Utilize a fórmula $S = \frac{4 \cdot Q}{\pi \cdot D^2} \cdot n$, considerando as seguintes condições:

- se $D > 100$, então $n = 2$;
- se $D < 50$, então $n = 6$;
- caso contrário, $n = 4$.

3.39. Altere o fluxograma da Figura 3.25 para calcular x^y, para quaisquer valores de x e y (incluindo 0).

3.40. Elabore um fluxograma que, dado um valor n inteiro, calculará seu fatorial. Lembrando, o fatorial de um número n é calculado pela expressão:

$$n! = n \cdot (n-1) \cdot (n-2) \cdot (n-3) \cdot \ldots 1$$

3.41. Elabore um fluxograma que deverá calcular o número de maneiras de se escolher r dentre n objetos diferentes, não importando a ordem. Lembrando:

$$C_{n,r} = \frac{n!}{r! \cdot (n-r)!}$$

em que n e r são valores digitados.

3.42. A fórmula de juros compostos é a seguinte:

$$V_f = (1+i)^N \cdot V_i$$

V_f é o valor final obtido após N períodos de aplicação com juros i. V_i é o valor inicial, à vista. Elabore um fluxograma que, após ler o valor inicial, o número de períodos (que normalmente são meses) e a taxa de juros, calcule o valor final desejado.

3.43. Escreva um fluxograma que leia três valores quaisquer para as variáveis A, B e C. A seguir, ordene esses valores, exibindo as mesmas variáveis A, B e C, agora já ordenadas.

3.44. Um número da série de Fibonacci é gerado a partir da soma de dois valores imediatamente anteriores. Convenciona-se que o primeiro número, f_0, é 0, e o segundo, f_1, é 1. A partir desses valores, é possível calcular o n-ésimo elemento da série assim (para $n > 2$):

$$f_n = f_{n-1} + f_{n-2}$$

Elabore um fluxograma que, a partir de um valor n lido ($n >= 0$), calcule f_n.

3.45. Para o fluxograma apresentado pela Figura 3.28, responda:

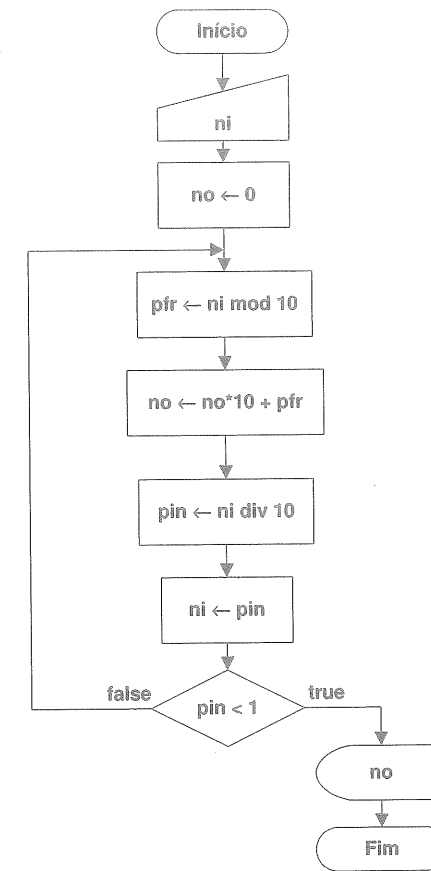



Figura 3.28 Fluxograma para o Exercício 3.45.

a) determine o valor de no para $ni = 8730$;

b) determine o valor de no para $ni = 1234$;

c) o que faz esse fluxograma?

3.46. Escreva um fluxograma que leia o nome do usuário e o cumprimente. Por exemplo, se você se chamar Aníbal, a resposta a ser exibida será: 'Olá Aníbal, meu nome é Chuck. Você quer brincar comigo?'

3.47. Elabore um fluxograma que permita a entrada de um número inteiro entre 1 e 9999 e escreva seu valor por extenso.

3.48. Elabore um fluxograma que compacte um texto contido em uma cadeia de caracteres, eliminando os seus espaços em branco. Mais especificamente, se o texto fornecido for 'O ␣ amor ␣ é ␣ lindo!' (␣ representa um espaço em branco), o resultado a ser apresentado deverá ser 'Oamorélindo!'.

3.49. Elabore um fluxograma que permita a entrada de N cadeias de caracteres e então exiba as seguintes estatísticas: o número total de caracteres digitados (incluindo espaços em branco) e o número total de palavras.

3.50. Elabore um fluxograma que permita a entrada de duas cadeias de caracteres, respectivamente, às variáveis $BUSCA$ e $MENSAGEM$, e então exiba todas as posições da cadeia contida em $BUSCA$ que foram localizadas em $MENSAGEM$.

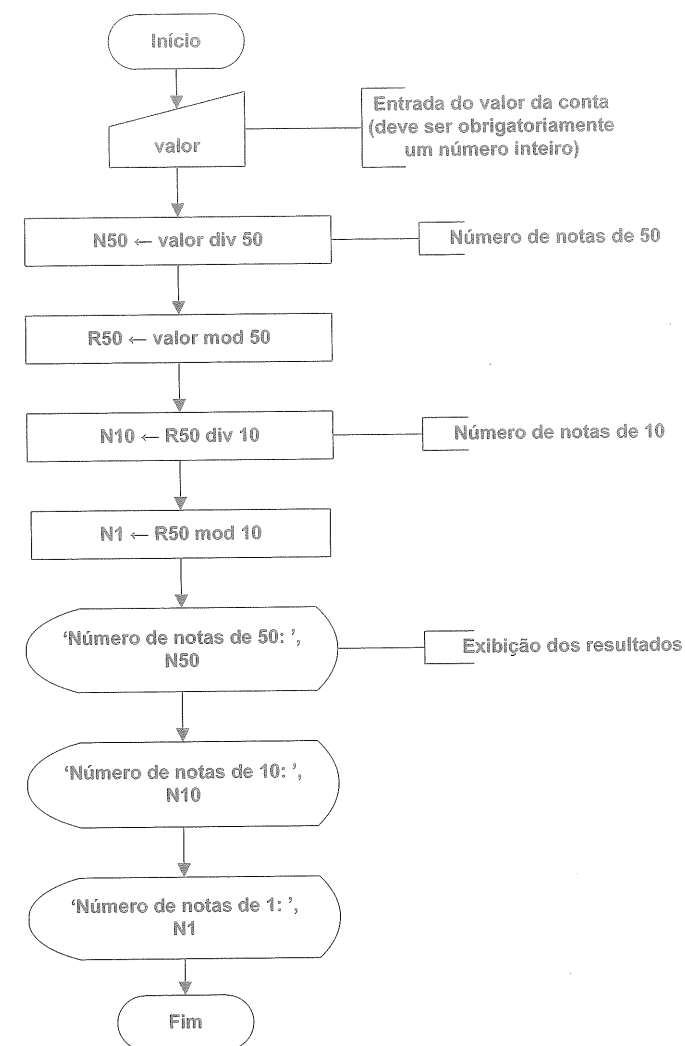
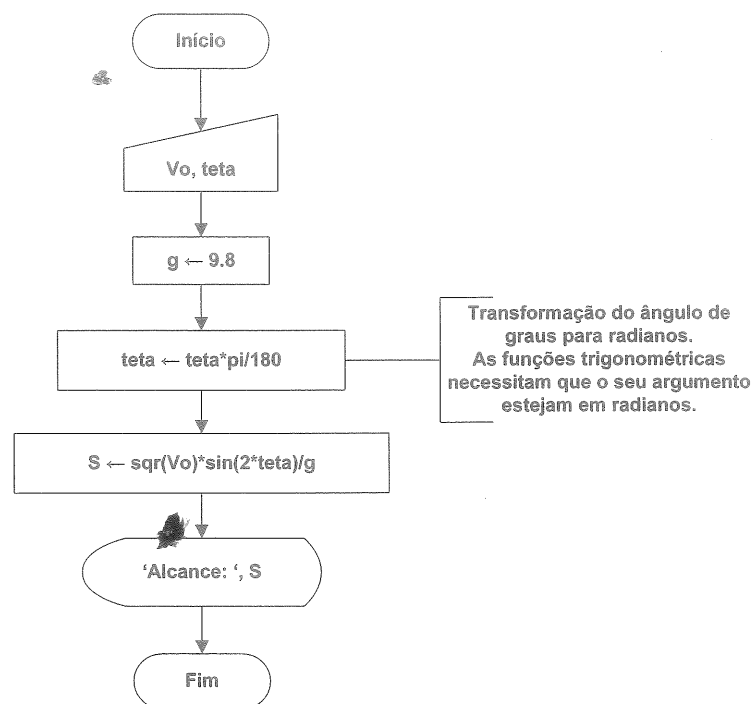
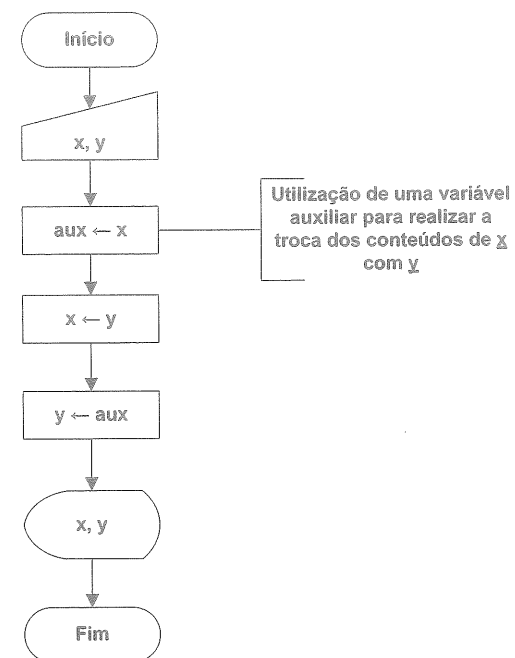
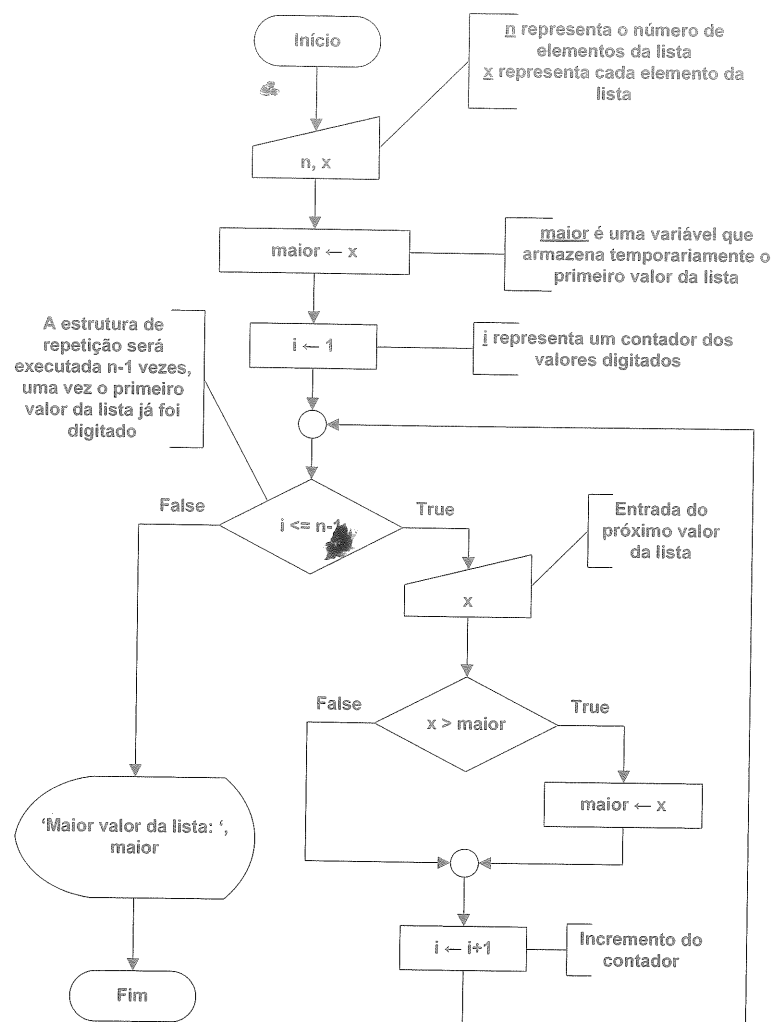
3.11 Exercícios resolvidos

3.1.

3.5. ☼

Início
Vo, teta
g ← 9.8
teta ← teta*pi/180
Transformação do ângulo de graus para radianos.
As funções trigonométricas necessitam que o seu argumento estejam em radianos.
S ← sqr(Vo)*sin(2*teta)/g
'Alcance: ', S
Fim

3.9. ☼

Início
x, y
aux ← x
Utilização de uma variável auxiliar para realizar a troca dos conteúdos de x com y
x ← y
y ← aux
x, y
Fim

3.11.

A solução apresentada permite encontrar o maior valor de uma lista contendo tanto números positivos quanto negativos.

Como ficaria a solução para encontrar o menor valor de uma lista?

O que aconteceria se a variável maior tivesse como valor inicial 0?

3.14.

3.23.

Início
n
i ← 0
i ← i+1
valor ← i*(i+1)*(i+2)
False
valor >= n
True
False
valor = n
True
n, 'não é triangular'
n, 'é triangular'
Fim

3.28.

Início
n
S ← 0
Valor inicial do acumulador
(elemento neutro da adição)
i ← 1
False
i <= n
True
'Soma: ',S
Fim
x
S ← S + sqr(x)
i ← i +1

3.30.

Início
A
S ← 0
i ← 1
den ← 3
A variável den representa o denominador da expressão
i <= 40
False
True
'Série: ',S
Fim
S ← S + 7*A/den
den ← den*2
i ← i +1

3.40.

Início
n
Fat ← 1
Valor inicial do acumulador (elemento neutro da multiplicação)
i ← 1
i <= n
False
True
'Fatorial: ', Fat
Fim
Fat ← Fat * i
i ← i + 1

3.48.

Capítulo 4

Estruturas de Programação

No Capítulo 3, foi apresentada uma representação gráfica de algoritmos denominada fluxograma, que utiliza um conjunto de símbolos da norma ISO 5807/1985. Aliado ao conhecimento de tipos básicos de dados, foram construídos fluxogramas com o intuito de serem futuramente implementados facilmente em qualquer linguagem de programação. O objetivo deste capítulo é formalizar as estruturas de programação já vistas, objetivando-se ficar mais próximo das estruturas que são encontradas nas linguagens de programação típicas, bem como apresentar nomes pelos quais essas estruturas são conhecidas no jargão da computação. Assim, neste capítulo, será feito um retrospecto do que foi apresentado anteriormente, classificando as estruturas de programação segundo os nomes pelos quais elas são habitualmente conhecidas e convencionando-se a forma de representá-las em fluxogramas. Por fim, serão apresentadas outras representações de algoritmos bem conhecidas: Portugol e diagramas de Nassi-Schneidermann.

4.1 Estruturas de programação

Como já foi apresentado no Capítulo 3, as instruções ou comandos utilizados em fluxogramas podem ser classificados como:

- Instruções sequenciais: representam ações imperativas, sem nenhum tipo de decisão.
- Instruções de decisão: representam um desvio no fluxo normal do algoritmo, conforme o resultado de uma expressão lógica.
- Instruções de repetição: representam a execução repetitiva de comandos existentes

em um desvio no fluxo normal de um programa, governada pelo resultado de uma expressão lógica.

Essas instruções formam o que se chama de **estruturas de programação**. São conhecidas respectivamente como **estruturas sequenciais**, **de decisão** e **de repetição**.

Conforme provado por Böhm e Jacopini em 1966, essas estruturas – também denominadas **estruturas primitivas de programação** – permitem a descrição de qualquer algoritmo que seja computável, sendo implementável em um computador. Em resumo, qualquer programa de computador pode ser escrito combinando-se esses três tipos de estruturas.

4.2 Estruturas sequenciais

As estruturas sequenciais de programação representam os comandos que são executados **imperativamente**, sem desvio algum de caminho. Os cálculos, a execução de funções e os procedimentos são exemplos dessas estruturas. Um fluxograma que contém apenas as estruturas sequenciais não apresenta **nenhum desvio em seu fluxo**.

Por exemplo, o fluxograma para calcular a força aplicada sobre a tampa de um tanque visto no Capítulo 3 é um exemplo de um fluxograma que somente emprega as estruturas sequenciais, revisto na Figura 4.1 a seguir:

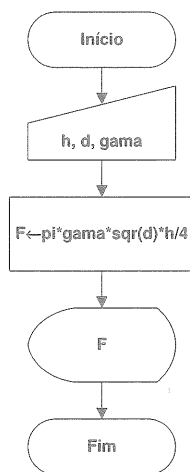



Figura 4.1 Exemplo de fluxograma com estruturas sequenciais.

4.3 Estruturas de decisão

São estruturas que permitem a tomada de uma decisão sobre qual o caminho a ser escolhido, de acordo com o resultado de uma *expressão lógica*. Existem três formas básicas desse tipo de estrutura: **SE-ENTÃO**, **SE-ENTÃO-SENÃO** e **CASO**.

4.3.1 Estrutura SE-ENTÃO

Essa estrutura é representada por um comando que avalia uma **expressão lógica**, **resultando** um valor que pode ser ***true*** ou ***false***. Como consequência desse resultado, o processamento se fará por um de dois caminhos: se o resultado for *true*, serão executados os comandos encontrados no caminho indicado pelo resultado *true*; caso contrário, será efetuado um desvio sem comando algum. Ambos os fluxos convergem para o final da estrutura. A estrutura SE-ENTÃO está representada na Figura 4.2.

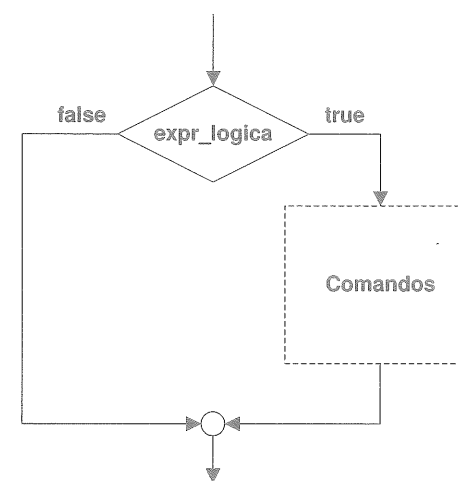



Figura 4.2 Estrutura de decisão SE-ENTÃO.

Na Figura 4.2, *expr_logica* representa alguma expressão lógica, que, se resultar *true*, vai permitir a execução de um conjunto de um ou mais comandos quaisquer, os quais podem ser sequenciais, de decisão ou de repetição. Se o resultado de *expr_logica* for *false*, nenhum comando será executado.

4.3.2 Estrutura SE-ENTÃO-SENÃO

Essa estrutura é representada por um comando que avalia uma expressão lógica, resultando um valor que pode ser *true* ou *false*. Graças a esse resultado, o processamento se fará por um de dois caminhos:

- se o resultado for *true*, serão executados os comandos encontrados no caminho indicado pelo resultado *true*;
- caso contrário, serão executados os comandos encontrados no caminho indicado pelo resultado *false*.

Nota-se que ambos os fluxos convergem para o final da estrutura. A estrutura SE-ENTÃO-SENÃO está representada na Figura 4.3.

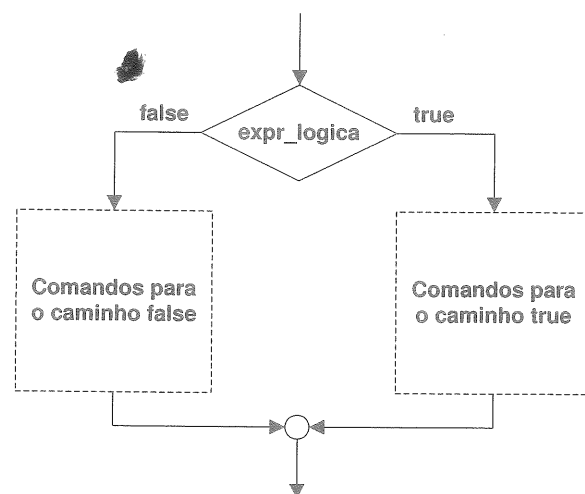



Figura 4.3 Estrutura de decisão SE-ENTÃO-SENÃO.

Na Figura 4.3, *expr_logica* representa alguma expressão lógica, que se resultar *true*, vai permitir a execução de um conjunto de um ou mais comandos quaisquer existentes no caminho *true*, os quais podem ser sequenciais, de decisão ou de repetição. Se o resultado de *expr_logica* for *false*, será executado um conjunto contendo um ou mais comandos quaisquer, existentes no caminho *false*, podendo, novamente, ter estruturas sequenciais, de decisão ou de repetição. Ambos os fluxos convergem para o final da estrutura.

4.3.3 Estrutura CASO

Nos dois tipos de estruturas apresentados nas Seções 4.3.1 e 4.3.2, ocorre uma escolha entre dois caminhos possíveis. A estrutura CASO possibilita escolher **mais de um caminho**, de acordo com um resultado a partir de uma *expressão inteira*. Aqui não se avalia uma expressão lógica, e, sim, uma expressão inteira, cujo resultado numérico vai determinar o caminho a ser seguido. Se nenhuma das opções for atendida, podemos definir um caminho-padrão. A estrutura CASO está representada na Figura 4.4.

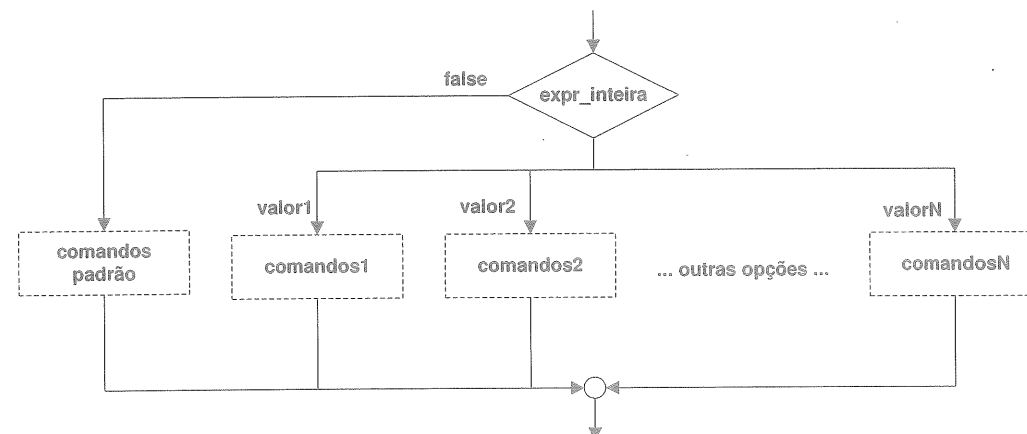



Figura 4.4 Estrutura de decisão CASO.

Na Figura 4.4, *expr_inteira* representa alguma expressão inteira, que, se resultar *valor*1, vai executar um conjunto de um ou mais comandos representados por *comandos*1. Se resultar *valor*2, vai executar um conjunto de um ou mais comandos representados por *comandos*2 e assim por diante até o último valor (*valorN*). Para esse caso, vai executar um conjunto de um ou mais comandos representados por *comandosN*.

Não existe nenhuma limitação do número de opções que podem ser definidas – isso dependerá do problema. Se o valor de *expr_inteira* não resultar em nenhum dos N valores predefinidos, especifica-se um **caso-padrão** que executa um conjunto de um ou mais comandos representados por *comandos-padrão*. O caso-padrão é rotulado com o valor *false* somente para indicar que será aquele a ser executado, caso nenhuma das alternativas anteriores sejam atendidas. O caso-padrão caracteriza-se por ser opcional nessa estrutura.

4.3.4 Exemplos de estruturas de decisão

Considera-se, novamente, aqui, o exemplo da equação de Bhaskara vista no Capítulo 3. Esse fluxograma está reescrito na Figura 4.5, com as estruturas SE-ENTÃO-SENÃO indicadas.

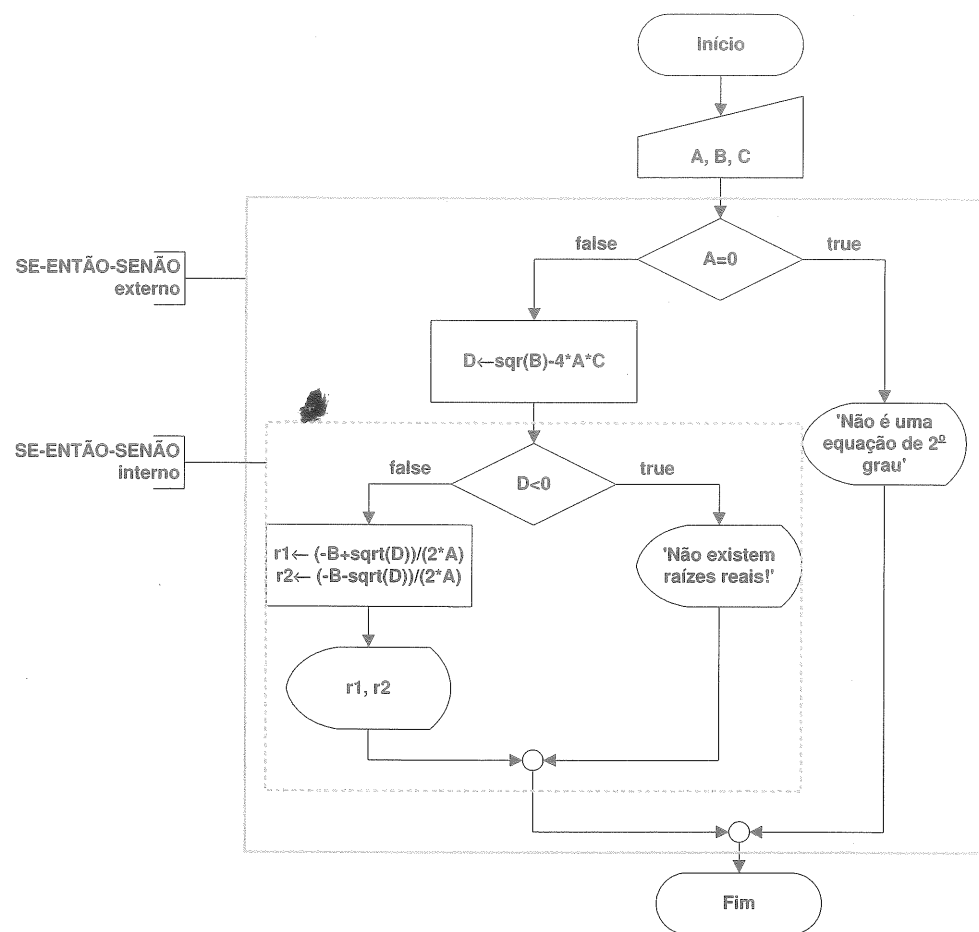



Figura 4.5 Exemplo de estrutura de decisão SE-ENTÃO-SENÃO.

A linha tracejada indica a estrutura SE-ENTÃO-SENÃO mais externa que verifica se $A = 0$. Se for $true$, então exibe-se a mensagem 'Não é uma equação de 2º grau'. Senão, calcula-se o valor de D e a seguir é executado mais uma estrutura SE-ENTÃO-SENÃO, para verificar se $D < 0$, indicada pela linha pontilhada. Se essa condição for $false$,

então se calculam e se exibem os valores de $r1$ e $r2$. Do contrário, exibe-se a mensagem 'Não existem raízes reais'.

Como um exemplo para a estrutura CASO, considere o seguinte problema: elaborar um fluxograma que simule uma calculadora simples, que some, subtraia, multiplique e divida um conjunto de números digitados. A ideia é digitar um número, um operador ('+', '−', '∗' ou '/') e outro número, sucessivamente até que se digite '=', quando o resultado for exibido.

Uma sugestão de fluxograma para resolver esse problema está representada na Figura 4.6.

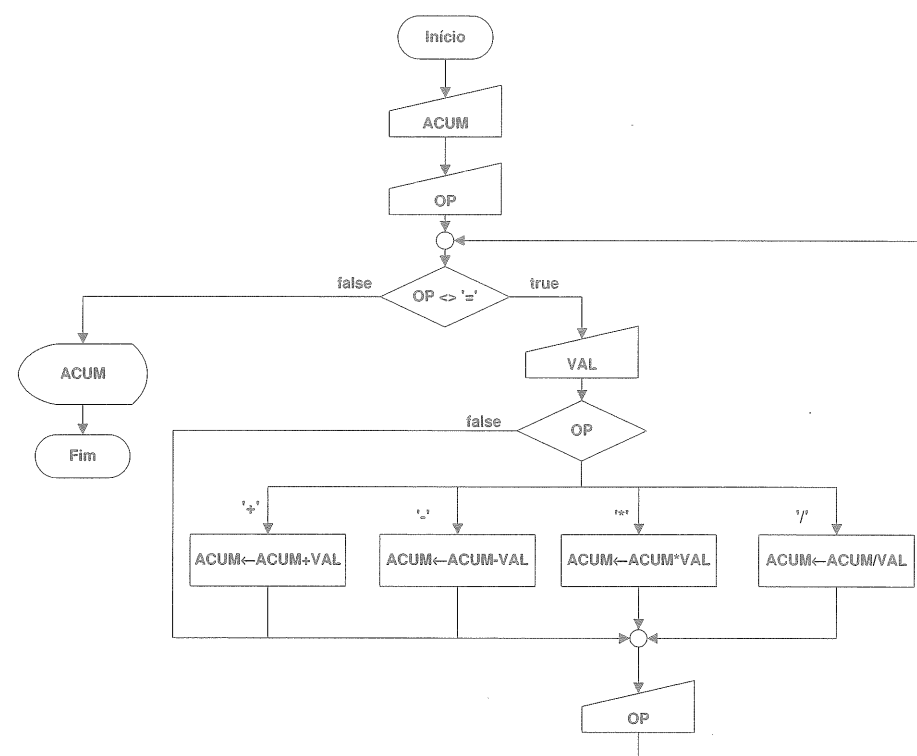



Figura 4.6 Exemplo de estrutura de decisão CASO.

Nesse fluxograma, as variáveis são:

- $ACUM$: é o acumulador das operações e contém o resultado a ser exibido, bem como representa o 1º operando das operações.

- VAL: representa os valores que serão digitados e fornecidos como o 2º operando das operações.
- OP: é uma variável do tipo CARACTERE e que contém o símbolo da operação a ser realizada ('+', '−', '∗' ou '/'). Os caracteres possuem um número inteiro associado (código ASCII).

Essa calculadora funciona assim: digita-se um primeiro valor ($ACUM$) e um operador (OP). O teste da repetição indicada verifica se o operador digitado é '='. Se for, exibe-se o conteúdo de $ACUM$. Senão, lê-se o segundo operando (VAL) e então decide-se, com uma estrutura CASO, qual operação será realizada. Assim:

- CASO OP seja '+', efetua-se uma soma.
- CASO OP seja '−', efetua-se uma subtração.
- CASO OP seja '∗', efetua-se uma multiplicação.
- CASO OP seja '/', efetua-se uma divisão.
- Nenhum dos casos acima: nada é executado (ignora-se o operador).

Depois, lê-se um novo operador e repete-se esse ciclo até que o operador seja '=', quando o valor de $ACUM$, representando o resultado, for apresentado.

4.4 Estruturas de repetição

São estruturas que permitem a repetição controlada de comandos. Podem ser dos tipos **ENQUANTO-FAÇA**, **REPITA-ATÉ** e **PARA-ATÉ-FAÇA**.

4.4.1 Estrutura ENQUANTO-FAÇA

A estrutura ENQUANTO-FAÇA permite a execução repetitiva de comandos ENQUANTO a condição de controle de repetição for $true$. Essa condição é uma expressão lógica da mesma forma que aquela que vimos em estruturas de decisão. A estrutura ENQUANTO-FAÇA está indicada na Figura 4.7.

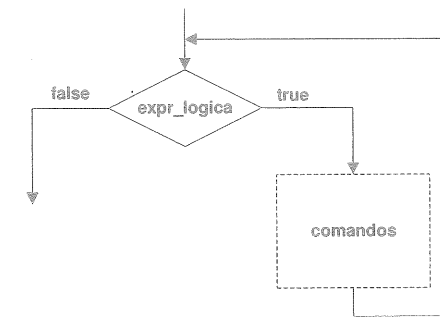



Figura 4.7 Estrutura de repetição ENQUANTO-FAÇA.

Nessa figura, $expr_logica$ representa alguma expressão lógica, que, enquanto resultar em $true$, vai permitir a execução repetitiva de comandos quaisquer representados por *comandos* (podem ser sequenciais, de decisão ou de repetição). Quando for $false$, segue-se para algum outro comando fora da repetição.

4.4.2 Estrutura REPITA-ATÉ

A estrutura REPITA-ATÉ possibilita a execução repetitiva de comandos até que a condição de controle de repetição seja $true$. Essa condição é uma expressão lógica da mesma forma que aquela que vimos em estruturas de decisão. A estrutura REPITA-ATÉ está indicada na Figura 4.8.

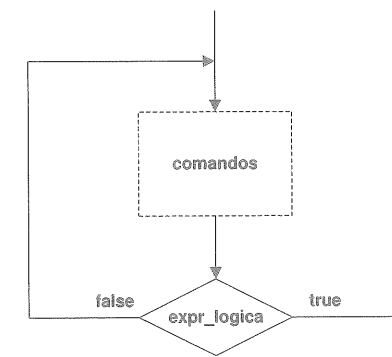



Figura 4.8 Estrutura de repetição REPITA-ATÉ.

Nessa figura, *expr_logica* representa alguma expressão lógica que, se resultar em *false*, vai permitir a repetição de comandos quaisquer representados por *comandos* (podem ser sequenciais, de decisão ou de repetição). Esses comandos são repetidos até que *expr_logica* seja *true*.

4.4.3 Estrutura PARA-ATÉ-FAÇA

Também conhecida como estrutura **DESDE-PARA-FAÇA**, é um caso particular da estrutura ENQUANTO-FAÇA. É particular, pois implementa uma estrutura ENQUANTO-FAÇA que vai repetir os comandos, utilizando-se de um contador que possui um certo valor inicial e que, por meio de incrementos unitários e inteiros (de 1 em 1), vai alcançar um valor final predefinido.

O número de repetições a serem executadas será função dos valores iniciais e finais do contador. Por ser um caso particular da estrutura ENQUANTO-FAÇA, essa estrutura é representada como na Figura 4.9.

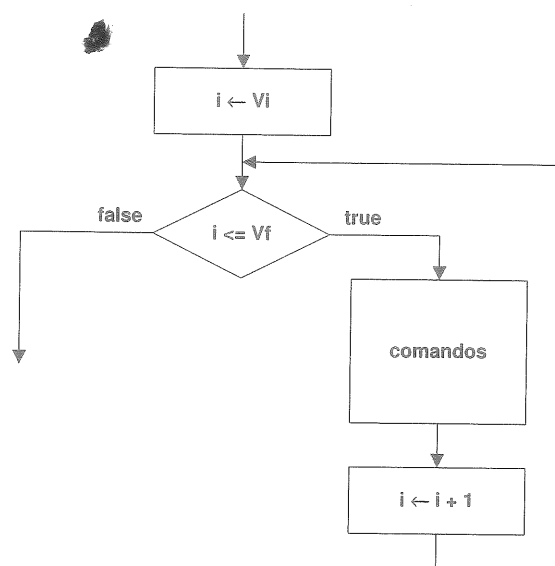



Figura 4.9 Estrutura de repetição PARA-ATÉ-FAÇA.

Nessa figura, i representa uma variável inteira (chamada variável de controle ou contadora), que será utilizada pelo comando PARA-ATÉ-FAÇA (DESDE-PARA-FAÇA), para executar um certo número de vezes as instruções quaisquer representadas por comandos (podem ser sequenciais, de decisão ou de repetição). Vi representa um número inteiro, ou variável inteira, indicando o valor inicial do contador e Vf representa um número inteiro, ou variável inteira, indicando o valor final do contador.

Por convenção, o incremento utilizado será sempre 1 e as repetições somente serão realizadas se $Vi <= Vf$. É proibido, dentro da estrutura PARA-ATÉ-FAÇA (DESDE--PARA-FAÇA), alterar o valor de sua variável contadora. Isso é obvio, já que, se isso for feito, não se estarão executando as repetições corretamente, e essa estrutura perderá seu propósito.

Por exemplo, o trecho de fluxograma da Figura 4.10 permite ler e somar N valores digitados, utilizando a estrutura PARA-ATÉ-FAÇA (DESDE-PARA-FAÇA).

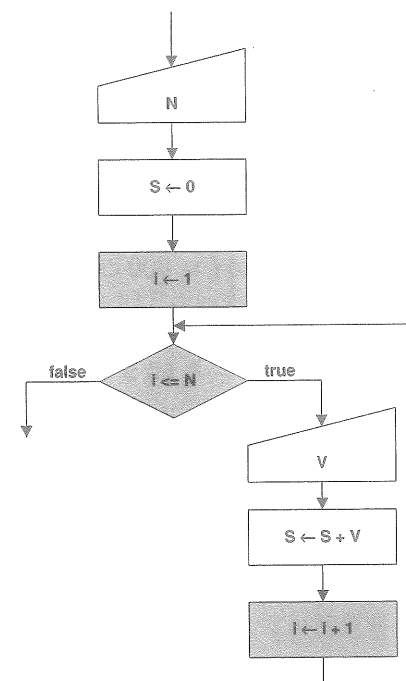



Figura 4.10 Identificação da estrutura de repetição PARA-ATÉ-FAÇA.

Aqui, os blocos que constituem a estrutura PARA-ATÉ-FAÇA (DESDE-PARA--FAÇA) estão em cinza, para a sua melhor identificação. Observa-se que o valor inicial da variável i é 1 e o final é N. Logo, a leitura de um valor se repete N vezes e depois realiza-se a soma deste com a anterior armazenada em S.

4.4.4 Exemplos de estruturas de repetição

Para exemplificar o uso de estruturas de repetição, será construído um algoritmo para calcular o ponto x no qual a *flecha* em uma viga é zero. A flecha é o deslocamento vertical que a viga sofre, quando submetida por forças aplicadas sobre ela, conforme a Figura 4.11.

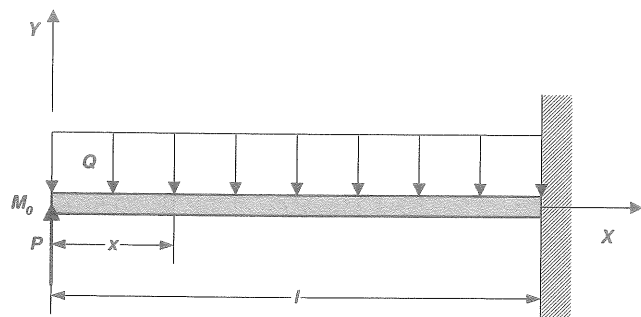



Figura 4.11 O problema do cálculo da flecha em uma viga.

Sabe-se que a viga possui comprimento l = 500 cm e que o enflechamento é obtido pela seguinte equação (medidas em cm):

$$y = -9,44 \times 10^{-10}x^4 + 7,55 \times 10^{-7}x^3 - 4,53 \times 10^{-6}x^2 - 8,99 \times 10^{-2}x + 10,7$$

Deseja-se encontrar qual é o ponto em que $x = 0$, ou seja, determinar a(s) raiz(raízes) da equação. O gráfico dessa equação é exibido na Figura 4.12, o que demonstra que ela possui uma única raiz no intervalo $[0, 500)$.

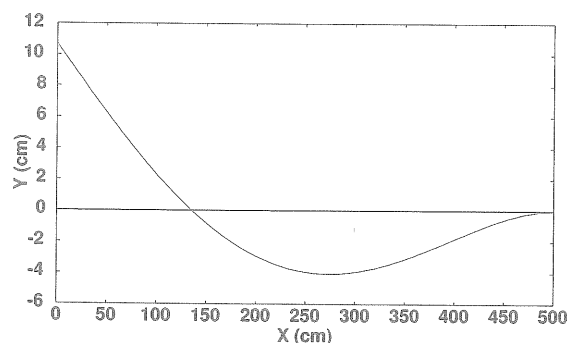



Figura 4.12 Enflechamento de uma viga.

Um algoritmo simples para a determinação das raízes reais de qualquer equação em um certo intervalo é conhecido como *método da bissecção*, descrito pelos seguintes passos:

1. Determinam-se dois valores, x_1 e x_2, para os quais os sinais de $f(x_1)$ e $f(x_2)$ sejam diferentes. Se a função $f(x)$ não possuir singularidades nesse intervalo, é garantida a existência de uma raiz em X, no intervalo $[x_1, x_2]$, conforme ilustrado na Figura 4.13.

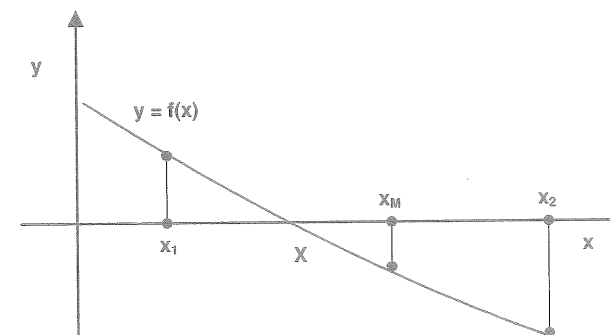



Figura 4.13 Bissecção de um intervalo.

2. Calcula-se o ponto $x_M = \frac{1}{2}(x_1 + x_2)$ e, a seguir, $f(x_M)$. Essa é uma primeira estimativa da raiz.

3. Se $f(x_1)$ e $f(x_M)$ possuírem sinais diferentes, substitui-se o valor de x_2 por x_M; se $f(x_2)$ e $f(x_M)$ têm sinais diferentes, substitui-se o valor de x_1 por x_M e, então, retorna-se ao passo 2. Se $f(x_M) = 0$ dentro de uma tolerância especificada, x_M será a raiz procurada.

Cabem aqui algumas explicações antes de se elaborar o algoritmo:

1. Para saber se dois números possuem o mesmo sinal, basta multiplicá-los entre si e verificar se o resultado é maior que zero.

2. A tolerância da raiz encontrada pode ser avaliada de acordo com a diferença absoluta entre os dois últimos valores estimados da raiz com a expressão $|x_i - x_{i-1}| \leq \epsilon$, em que ϵ é o erro do cálculo da raiz, que deve ser imposto na solução.

Voltando ao problema da viga, a ideia é utilizar o método da bissecção para se determinar o ponto x no qual a flecha indica zero. Na elaboração do algoritmo para resolver esse problema, serão utilizadas como entrada e saída as seguintes variáveis:

- Entrada: a e b para armazenar o intervalo inicial no qual se procura a raiz e e para armazenar o erro de cálculo admitido.
- Saída: x_M, a raiz procurada.

A seguir, tem-se a implementação da solução desse problema, utilizando as estruturas de repetição apresentadas.

Utilizando a estrutura ENQUANTO-FAÇA

A versão do fluxograma para esse problema, utilizando a estrutura ENQUANTO-FAÇA, está apresentada na Figura 4.14. Nessa figura, a estrutura ENQUANTO-FAÇA está destacada, bem como as de decisão SE-ENTÃO que são internas à repetição.

São utilizadas as seguintes variáveis adicionais:

- x: armazena o valor de x_M calculado em uma iteração anterior (inicializada, portanto, com zero);
- fa, fb e fm: armazenam o valor da função do problema nos pontos a, b e x_M descritos anteriormente.

Observa-se ainda que, para o cálculo da função do problema, foram utilizadas as funções matemáticas apresentadas na Tabela 3.9 do Capítulo 3.

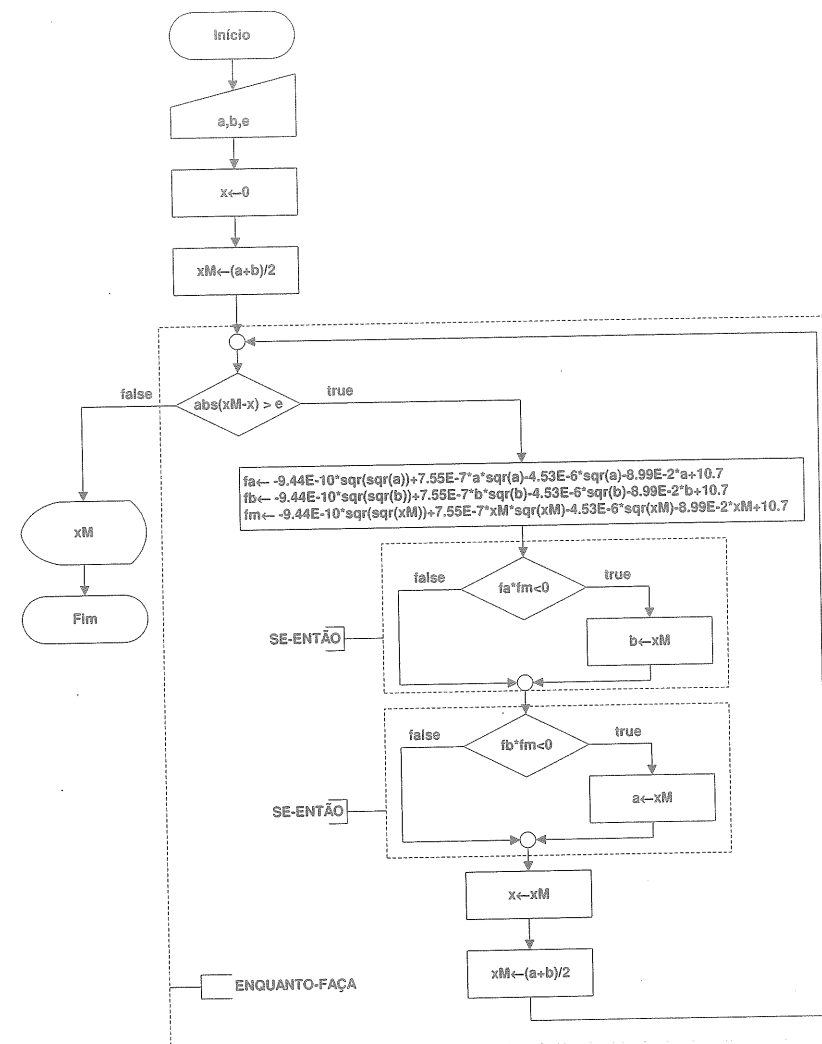



Figura 4.14 Exemplo da estrutura de repetição ENQUANTO-FAÇA.

Utilizando a estrutura REPITA-ATÉ

A versão do fluxograma para o problema, utilizando a estrutura REPITA-ATÉ, está mostrada na Figura 4.15.

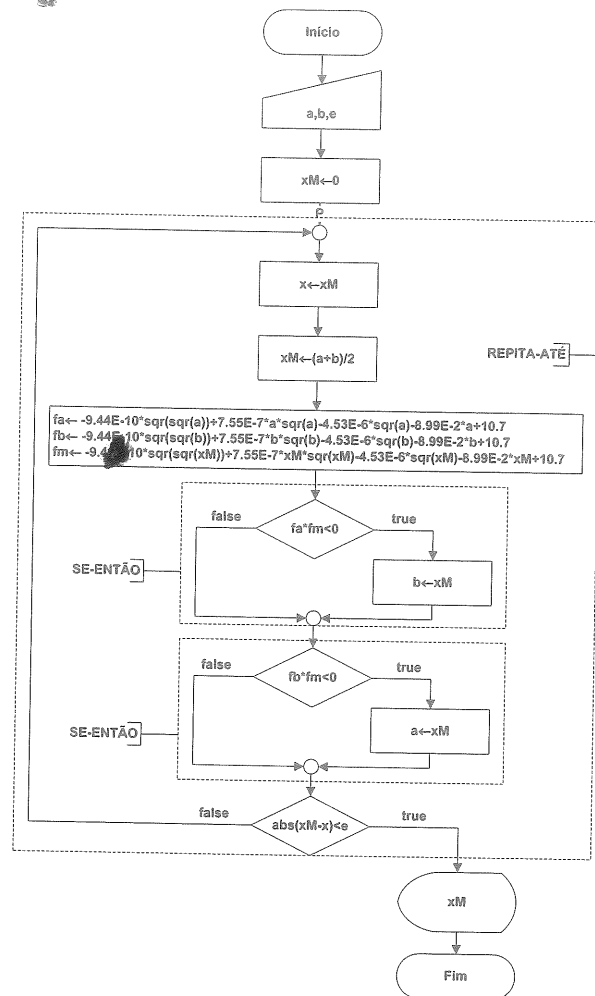



Figura 4.15 Exemplo da estrutura de repetição REPITA-ATÉ.

Nessa figura, a estrutura REPITA-ATÉ está destacada, bem como aquelas de decisão SE-ENTÃO que são internas à repetição. São utilizadas as mesmas variáveis da implementação com ENQUANTO-FAÇA e valem ainda as mesmas observações.

Utilizando a estrutura PARA-ATÉ-FAÇA

Para a versão do fluxograma utilizando a estrutura PARA-ATÉ-FAÇA, é necessário observar primeiro que essa estrutura se baseia em um contador inteiro para determinar o número de execuções a realizar.

No algoritmo proposto, verifica-se que o número de repetições é dependente dos valores de intervalo e de erro fornecidos. Além disso, nos fluxogramas anteriores, a condição de parada da repetição envolvia as expressões reais, o que não será possível neste caso.

Analisando o algoritmo proposto, ele basicamente subdivide uma região real em partes idênticas e continua realizando essa operação até que o critério de erro seja atendido. Cada novo intervalo é dividido em dois subintervalos, portanto.

O número de subintervalos a considerar seria, no mínimo, $(b - a)/e$, isto é, o menor subintervalo a considerar teria o mesmo tamanho do erro e. Assim, o processo de divisão de intervalos proposto por esse algoritmo poderia fornecer uma 'árvore' conforme a Figura 4.16.

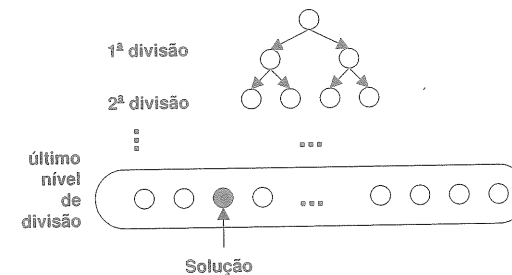



Figura 4.16 Processo de divisão utilizado pelo algoritmo da bissecção.

Como o algoritmo divide sempre um intervalo por 2, a solução final estaria em um nível da árvore contendo $(b - a)/e$ intervalos. A altura da árvore determina, portanto, o número de divisões que o algoritmo deverá executar. Por ser uma árvore binária (cada elemento gera apenas dois elementos), a altura da árvore h é calculada de acordo com o número de elementos que existem em seu nível mais baixo, n, dessa forma: $h = \log_2 n$.

Voltando ao problema, para a implementação com a estrutura PARA-ATÉ-FAÇA, basta adicionar uma variável contadora i e fazer a repetição dos comandos que dividem o intervalo e decidem qual será o novo intervalo a ser feito de 1 até $\log_2 (b - a)/e$. Assim, o fluxograma que representa a solução utilizando o comando PARA-ATÉ-FAÇA está apresentado na Figura 4.17. Nesse fluxograma são utilizadas outras funções matemáticas mostradas na Tabela 3.9 do Capítulo 3.

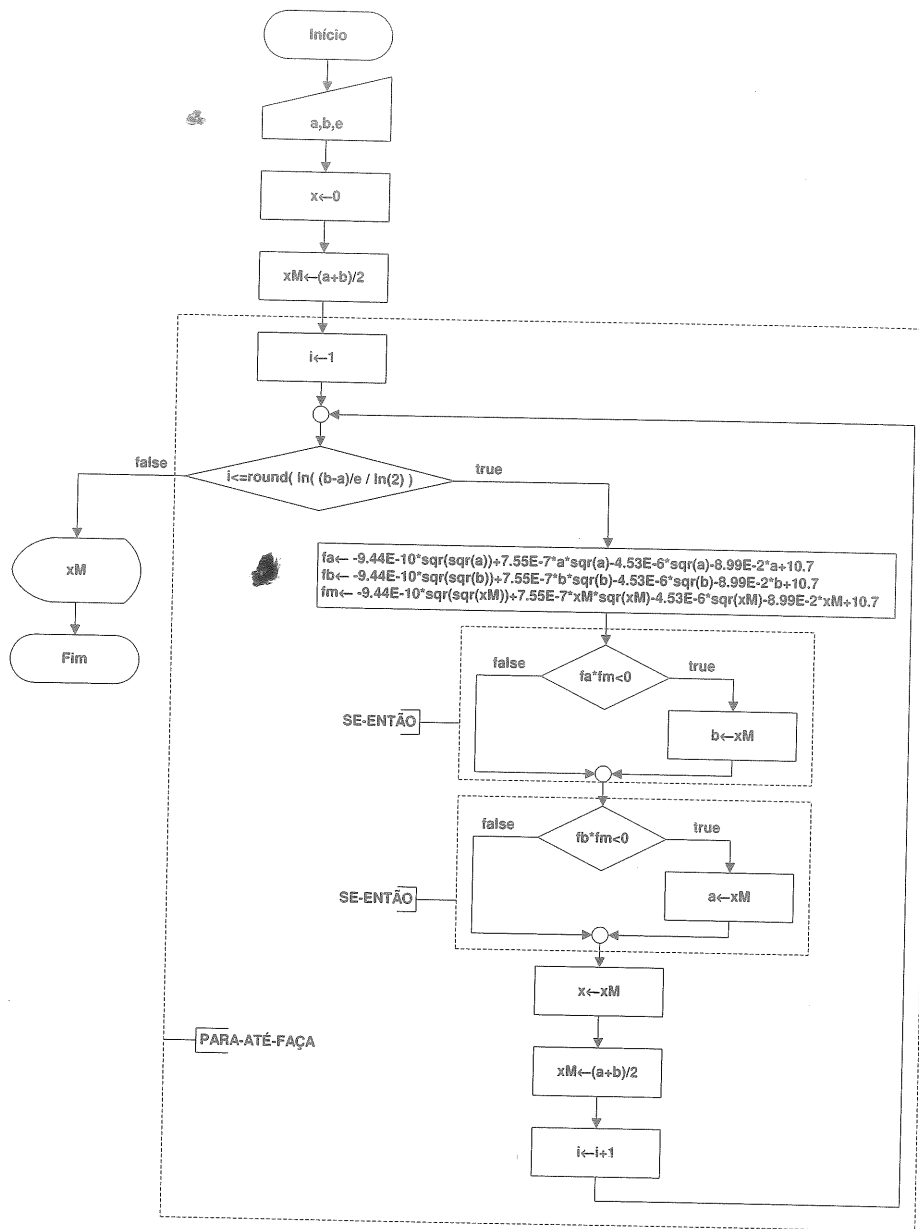



Figura 4.17 Exemplo da estrutura de repetição PARA-ATÉ-FAÇA.

4.4.5 Símbolos específicos para estruturas de repetição (ISO 5807)

Existe um símbolo da norma ISO[1] 5807/1985, que será adotado neste livro, específico para executar repetições em que a forma da estrutura é única para qualquer caso. Nesse símbolo indica-se o tipo de repetição que será executado, anotando em seu interior as expressões que devem ser obedecidas. Esse símbolo é escrito de acordo com a Figura 4.18.

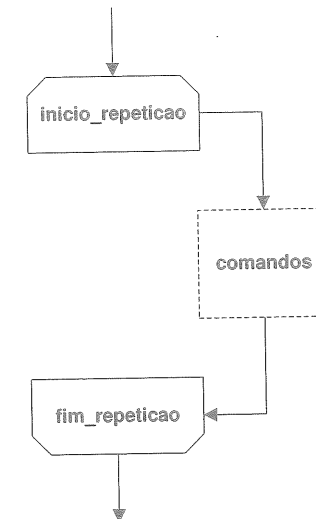



Figura 4.18 Símbolo específico para as estruturas de repetição (ISO 5807).

Nessa figura, *inicio_repeticao* e *fim_repeticao* são anotações que devem ser feitas para indicar qual será o tipo da estrutura de repetição desejada (ENQUANTO-FAÇA, REPITA-ATÉ e PARA-ATÉ-FAÇA ou DESDE-PARA-FAÇA). Os comandos a serem repetidos ficam entre os símbolos que marcam o início e o fim da repetição e subentende-se que a repetição será executada de acordo com o que foi definido no interior dos símbolos delimitadores.

O uso desse símbolo para as três estruturas de repetição vistas anteriormente é feito de acordo com a Figura 4.19. Nota-se que a estrutura é a mesma. O que diferencia cada tipo de repetição são as anotações feitas no interior dos símbolos de início e fim de repetição.

[1]International Organization for Standardization.

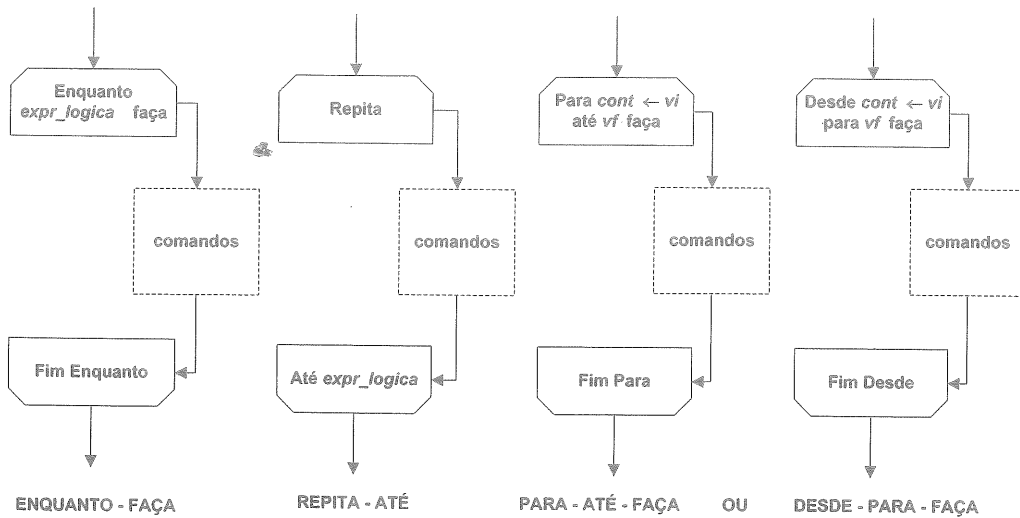



Figura 4.19 Uso do símbolo específico para as estruturas de repetição.

Na representação da estrutura PARA-ATÉ-FAÇA (DESDE-PARA-FAÇA) está subentendido que o contador é automaticamente incrementado e controlado (a partir do valor inicial, alcança-se o valor final – inclusive – em incrementos unitários). É dessa forma que o comando PARA-ATÉ-FAÇA (DESDE-PARA-FAÇA) é implementado na maioria das linguagens de programação.

Como exemplo do uso desse símbolo, são apresentadas na Figura 4.20 três versões de trechos de fluxogramas que permitem ler e somar N valores digitados.

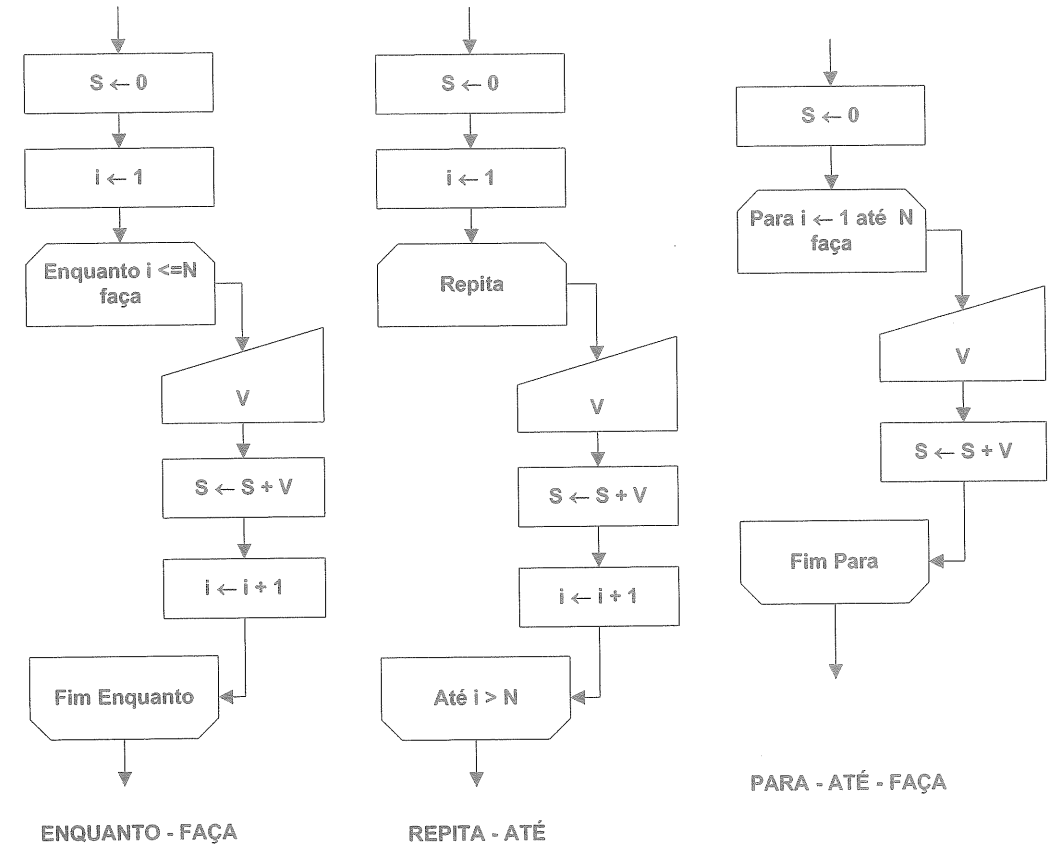



Figura 4.20 Exemplos de uso do símbolo específico para as estruturas de repetição.

4.5 Outras representações de algoritmos

4.5.1 Portugol

O Portugol ou português estruturado é uma técnica textual de representação de algoritmos na qual as estruturas de programação são representadas por um subconjunto de palavras da língua portuguesa. As características básicas do Portugol serão mostradas a seguir por meio de exemplos.

Algoritmo mínimo

Um algoritmo em Portugol é delimitado pelas palavras *Início* e *Fim*, segundo o Algoritmo 4.1.

Algoritmo 4.1 Algoritmo mínimo em Portugol.

Início
Fim

Algoritmo com instruções sequenciais

As instruções sequenciais, como as atribuições de variáveis, as expressões e o uso de sub-rotinas, são escritas como em fluxogramas, de acordo o Algoritmo 4.2.

Algoritmo 4.2 Algoritmo em Portugol com instruções sequenciais.

Início
  $x \leftarrow x + 1$
  $t \leftarrow sin(0.23)$
Fim

O deslocamento para a direita é proposital. Esse deslocamento ou endentação permite uma leitura mais fácil do algoritmo.

Algoritmo com comandos para a leitura ou exibição

Os comandos para a leitura e exibição de valores são representados pelas rotinas *Leia* e *Exiba*, de acordo com o Algoritmo 4.3.

Algoritmo 4.3 Algoritmo em Portugol com comandos de leitura e exibição.

Início
  $Leia(d, h, gama)$
  $F \leftarrow 3.1415 * gama * srq(d)/4$
  $Exiba(F)$
Fim

Tanto em *Leia* quanto em *Exiba*, as variáveis (e/ou constantes no caso de *Exiba*) são separadas por vírgula.

Algoritmo com estruturas de decisão

As estruturas condicionais SE-ENTÃO e SE-ENTÃO-SENÃO são escritas segundo o Algoritmo 4.4.

Algoritmo 4.4 Algoritmo em Portugol com estruturas de decisão.

Início
  $Leia(x)$
  $y \leftarrow 0$
  Se $x > 0$ **Então**
    $y \leftarrow x + 1$
  Fim Se
  Se $y > 0$ **Então**
    $z \leftarrow y + 3$
  Senão
    $z \leftarrow y + 2$
  Fim Se
  $Exiba(z)$
Fim

Finaliza-se a estrutura **Se** com **Fim Se** para não haver confusões. Isso é válido para os dois casos.

Algoritmo com estrutura de repetição ENQUANTO-FAÇA

A estrutura repetitiva ENQUANTO-FAÇA é escrita de acordo com o Algoritmo 4.5.

Algoritmo 4.5 Algoritmo em Portugol com estrutura ENQUANTO-FAÇA.

Início
  $Leia(N)$
  $S \leftarrow 0$
  $i \leftarrow 1$
  Enquanto $i <= N$ **Faça**
    $Leia(V)$
    $S \leftarrow S + V$
    $i \leftarrow i + 1$
  Fim Enquanto
  $Exiba(S)$
Fim

Conclui-se a estrutura ENQUANTO-FAÇA com **Fim Enquanto** para não haver confusões!

Algoritmo com estrutura de repetição REPITA-ATÉ

A estrutura de repetição REPITA-ATÉ é escrita conforme o Algoritmo 4.6.

Algoritmo 4.6 Algoritmo em Portugol com estrutura REPITA-ATÉ.

Início
  $Leia(N)$
  $S \leftarrow 0$
  $i \leftarrow 1$
  Repita
    $Leia(V)$
    $S \leftarrow S + V$
    $i \leftarrow i + 1$
  Até $i > N$
  $Exiba(S)$
Fim

A estrutura REPITA-ATÉ é finalizada pela palavra **Até**.

Algoritmo com estrutura de repetição PARA-ATÉ-FAÇA

A estrutura de repetição PARA-ATÉ-FAÇA (DESDE-PARA-FAÇA) é escrita consoante os Algoritmos 4.7 e 4.8.

Algoritmo 4.7 Algoritmo em Portugol com estrutura PARA-ATÉ-FAÇA.

Início
  $Leia(N)$
  $S \leftarrow 0$
  Para $i \leftarrow 1$ **Até** N **Faça**
    $Leia(V)$
    $S \leftarrow S + V$
  Fim Para
  $Exiba(S)$
Fim

Algoritmo 4.8 Algoritmo em Portugol com estrutura DESDE-PARA-FAÇA.

Início
  $Leia(N)$
  $S \leftarrow 0$
  Desde $i \leftarrow 1$ **Para** N **Faça**
    $Leia(V)$
    $S \leftarrow S + V$
  Fim Desde
  $Exiba(S)$
Fim

Finaliza-se a estrutura PARA-ATÉ-FAÇA (DESDE-PARA-FAÇA) com **Fim Para** (**Fim Desde**) para não haver confusões!

4.5.2 Diagramas de Nassi-Schneidermann

São diagramas que representam o algoritmo por uma "grande caixa" cujo interior é subdividido de forma conveniente a permitir o fácil entendimento do algoritmo. Essa "grande caixa" pode ocupar uma folha inteira ou parte dela.

Algoritmo com instruções sequenciais

Um algoritmo com instruções sequenciais é representado por subdivisões retangulares internas como mostrado na Figura 4.21, cada uma significando um comando.

Leia(d, gama, h)
F ← 3.1415*gama*sqr(d)/4
Exiba(F)

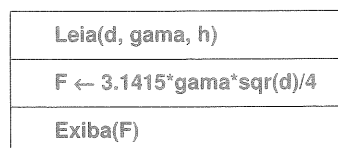



Figura 4.21 Algoritmo em Nassi-Schneidermann com instruções sequenciais.

Algoritmo com estruturas de decisão

As estruturas condicionais são representadas conforme a Figura 4.22, tomando como exemplo a resolução da equação de 2º grau por Bhaskara. Observa-se a bifurcação do teste.

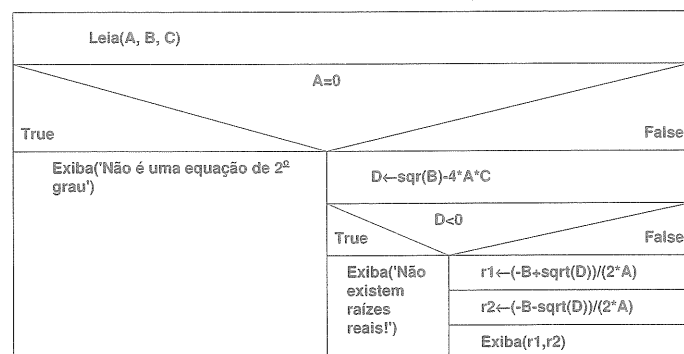



Figura 4.22 Algoritmo em Nassi-Schneidermann com estruturas de decisão.

Algoritmo com estrutura de repetição ENQUANTO-FAÇA

A estrutura de repetição ENQUANTO-FAÇA é ilustrada na Figura 4.23, tomando como exemplo a leitura e a soma de N elementos. O "L" invertido abriga as instruções a serem repetidas.

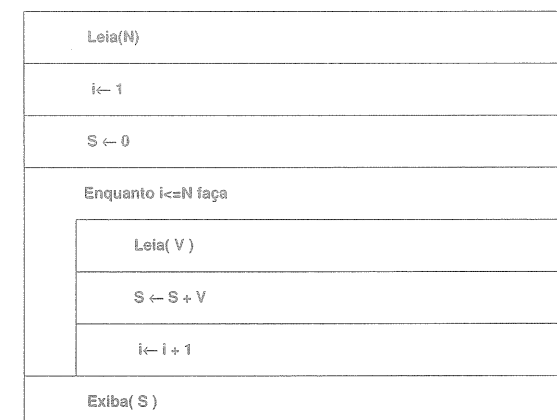



Figura 4.23 Algoritmo em Nassi-Schneidermann com a estrutura ENQUANTO-FAÇA.

Algoritmo com estrutura de repetição REPITA-ATÉ

A estrutura de repetição REPITA-ATÉ é apresentada na Figura 4.24, tomando como exemplo a leitura e a soma de N elementos. O "L" abriga as instruções a serem repetidas. Não é necessário digitar a palavra **repita**.

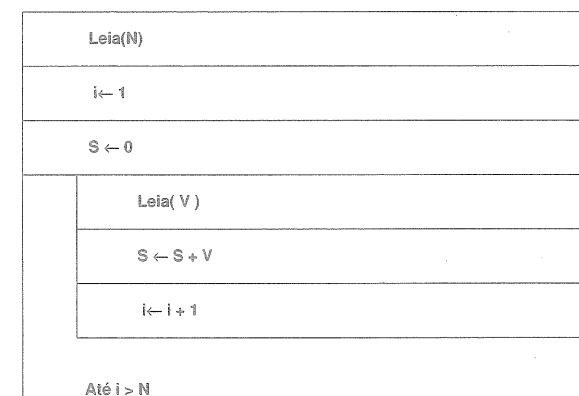



Figura 4.24 Algoritmo em Nassi-Schneidermann com a estrutura REPITA-ATÉ.

4.6 Exercícios

4.1. ☼ Considere o fluxograma da Figura 4.25.

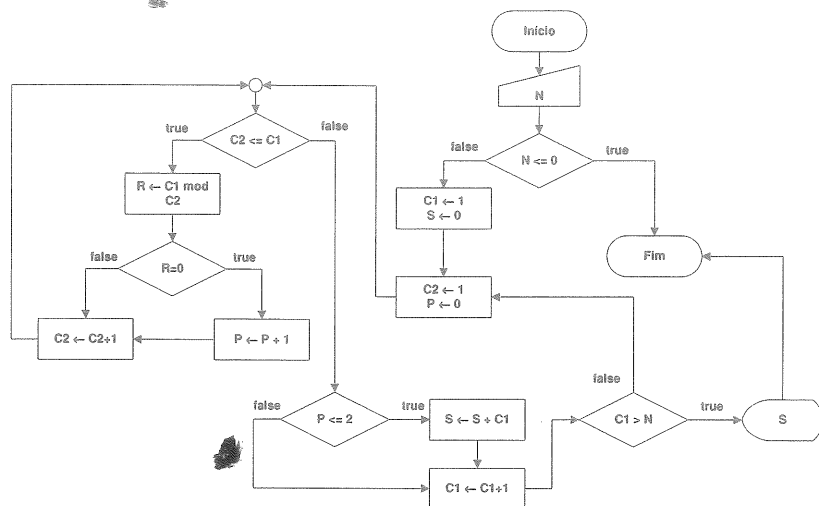



Figura 4.25 Fluxograma do Exercício 4.1.

Pede-se:

a) Reescrever esse fluxograma de acordo com as convenções vistas neste capítulo e de modo que se torne inteligível.

b) Identificar as estruturas de programação nele contidas.

c) Para que serve esse fluxograma? Simule-o para os seguintes valores de N: 1, 2, 3 e 7.

4.2. ☼ Considere o fluxograma da Figura 4.26.

Responda:

a) Dentre as três estruturas de repetição vistas neste capítulo, qual delas foi utilizada no diagrama de blocos apresentado?

b) Reescreva o diagrama apresentado (fazendo as adaptações necessárias) para as outras duas estruturas de repetição conhecidas.

c) O que aconteceria se não fosse colocado o incremento da variável de controle na questão anterior? Qual falha ocorrerá ao se testar o diagrama (teste de mesa)?

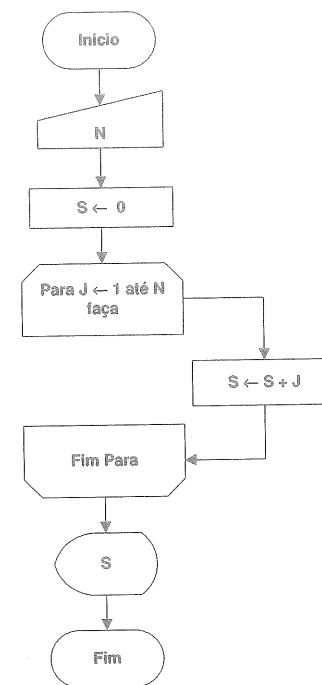



Figura 4.26 Fluxograma do Exercício 4.2.

4.3. ☼ Elabore um fluxograma que permita a entrada de uma hora de início e uma hora de término de uma palestra e que calcule sua duração, exibindo a quantidade de horas e minutos.

4.4. ☼ Produza um fluxograma que permita a entrada de N números quaisquer e que exiba a quantidade de números negativos. Escreva esse fluxograma com as estruturas:

a) ENQUANTO-FAÇA;

b) REPITA-ATÉ;

c) PARA-ATÉ-FAÇA.

4.5. ☼ ✎ Realize um fluxograma que possibilite a entrada de N valores de nomes e salários e que exiba como resultado o salário médio calculado e o nome da pessoa que recebe o maior salário. Escreva esse fluxograma com as estruturas:

a) ENQUANTO-FAÇA;

b) REPITA-ATÉ;

c) PARA-ATÉ-FAÇA.

4.6. Deseja-se construir um fluxograma para projetar futuramente um programa de auxílio a uma eleição. Os votos válidos são representados pelos números 1, 2 e 3, cada um correspondendo a um candidato. O voto em branco é representado pelo número 0 e o voto nulo, pelo número -1. Esse fluxograma deverá processar N respostas da votação. O fluxograma deverá calcular e exibir:

a) o total de votos para cada candidato;

b) o total de votos em branco;

c) o total de votos nulos;

d) o número do candidato vencedor (ou indicar se não houve vencedor, caso a população tenha anulado ou deixado em branco todos os votos).

4.7. Escreva um fluxograma que exiba o triângulo de Pascal, conforme indicado a seguir:

```
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
⋮ ⋮ ⋮ ⋮ ⋮ ⋮
```

4.8. Traduza o fluxograma da Figura 4.14 para:

a) Portugol;

b) Nassi-Schneidermann.

4.9. Reescreva o fluxograma do Exercício 3.33, utilizando as estruturas de repetição:

a) ENQUANTO-FAÇA;

b) REPITA-ATÉ;

c) PARA-ATÉ-FAÇA.

4.10. Escreva um fluxograma que permita a entrada de um número N inteiro e então exiba a decomposição desse número em seus fatores primos, assim:

$$6 = 2(1)\,3(1)$$
$$9 = 3(2)$$
$$24 = 2(3)\,3(1)$$

O número entre parênteses indica a potência do fator.

4.11. Escreva um fluxograma que gere os N primeiros números perfeitos. Um número perfeito é aquele que é igual à soma dos seus divisores, por exemplo, $6 = 1+2+3$.

4.12. Utilizando os resultados do Exercício 7, escreva a expansão da expressão $(a+b)^n$, para um valor de n lido. Os termos da expansão são os valores da n-ésima linha do triângulo de Pascal, por exemplo:

$$(a+b)^3 = a^3 + 3a^2b + 3ab^2 + b^3$$

O fluxograma deverá exibir a resposta no seguinte formato:

$$(a+b)\hat{}3 = a\hat{}3 + 3*a\hat{}2*b + 3*a*b\hat{}2 + b\hat{}3$$

Em que $*$ representará a operação de multiplicação e $\hat{}$, a operação de potenciação. O tipo de dado a ser exibido é uma cadeia de caracteres.

4.13. Escreva um fluxograma que, dada uma cadeia de caracteres S, vai exibir se essa cadeia contém um número inteiro positivo válido. O número inteiro válido a ser considerado deve conter apenas os caracteres '0', '1', '2', '3', '4', '5', '6', '7', '8' e '9'.

4.14. O mesmo do Exercício 13, agora incluindo números inteiros negativos.

4.15. Seguindo o estilo do Exercício 13, escreva um fluxograma que, dada uma cadeia de caracteres S, vai exibir se essa cadeia contém um número real válido. Considere que o número real poderá ser positivo ou negativo e que o separador decimal será o símbolo '.' (quando houver).

4.16. Um sistema de cargas possui um robô cujo braço é um garfo utilizado para mover as caixas de produtos que vêm de uma esteira de um setor de cargas para dentro de um caminhão. O robô é fixo e somente pode girar em seu eixo.

Seu braço pode ser levantado ou abaixado, pode avançar ou recuar para respectivamente encaixar ou soltar uma caixa. O robô apenas pode girar se seu braço estiver recuado (veja a Figura 4.27).

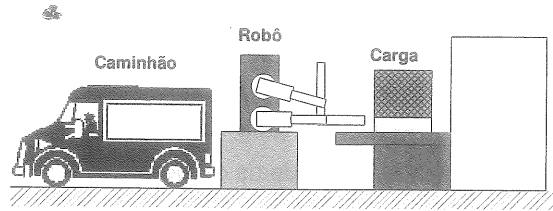



Figura 4.27 Figura do Exercício 4.16.

Escreva um fluxograma que permita ao robô mover um conjunto de caixas da esteira para o caminhão, utilizando o conjunto de instruções da Tabela 4.1, existentes na biblioteca de instruções do robô:

Tabela 4.1 Tabela para o Exercício 4.16.

Instrução	Descrição
POSICAO	Testa para verificar qual é a posição do braço do robô. Se o resultado for 0, o braço está ao lado da plataforma de cargas; se for 1, o braço está ao lado do caminhão.
GIRA_PARA(X)	Executa a ação de girar o seu braço. Se X for 0, o braço é girado para a plataforma de cargas; se X for 1, o braço é girado para o caminhão.
ESTA_VAZIO	Testa se o garfo do braço do robô está vazio ou não. O robô somente pode carregar uma caixa se o seu braço estiver vazio. Devolve valores *true* ou *false*.
ABAIXAR_BRACO	Executa a ação de abaixar o braço do robô. O robô apenas pode pegar ou soltar uma caixa se abaixar seu braço.
LEVANTAR_BRACO	Executa a ação de levantar o braço do robô. O robô somente pode mover uma caixa se levantar seu braço.
EXISTE_CARGA	Testa se existe alguma carga no terminal. Devolve valores *true* ou *false*.
AVANCAR_BRACO	Avança o braço do robô para pegar uma carga ou para soltar uma carga.
RECUAR_BRACO	Recua o braço do robô para soltar uma carga e para poder girar.
CAMINHAO_OK	Verifica se o caminhão está parado na plataforma aguardando por cargas. Devolve valores *true* ou *false*.

A posição inicial e final do braço do robô deve ser sempre **ao lado da esteira de cargas, recuado** e **vazio**.

4.17. Escreva um fluxograma que permita a entrada de uma cadeia de caracteres S, e então escreva as possíveis rotações à esquerda dessa cadeia. Por exemplo, se for digitada a cadeia 'Banana', deverá ser exibido:

'Banana'

'ananaB'

'nanaBa'

'anaBan'

'naBana'

'aBanan'

'Banana'

4.18. Elabore um fluxograma para fazer um pequeno robô em forma de seta percorrer a roseta em espiral quadrada (veja a Figura 4.28). A roseta é descrita a seguir como o caminho que liga o ponto A ao ponto B. Cada quadradinho representa uma unidade de deslocamento. O robô executa, por meio de seus microcontroladores, apenas três processos:

- $Desloca(X)$: desloca o robo em X unidades para a frente. X pode ser $1, 2, 3, \ldots$ de deslocamento na direção da seta.
- $ViraDireita$: apenas gira o robô para a direita.
- $ViraEsquerda$: somente gira o robô para a esquerda.

4.19. Um comitê olímpico solicitou a elaboração de um fluxograma para atender às competições de natação que serão realizadas em um clube. Como entrada de valores, esse fluxograma deverá receber o número de competidores (N) e os seus respectivos tempos (em segundos).

Como resultado, o fluxograma deverá apresentar o tempo médio obtido, levando-se em consideração todos os nadadores. Também deve exibir o melhor e o pior tempo conseguidos na competição.

4.20. Reescreva o fluxograma da Figura 4.6, para que a calculadora seja operada de forma *pós-fixa*, ou seja, digitam-se primeiro os operandos e depois a operação. Por exemplo, a operação $3 + 3 - 2$, que resulta em 4, é escrita assim: $3\ 3 + 2 -$.

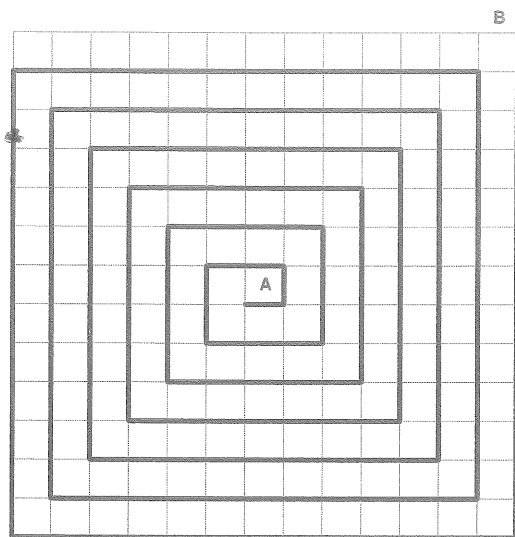



Figura 4.28 Desenho da roseta do Exercício 4.18.

Prove que essa versão permitirá o uso iterativo da calculadora sem a necessidade alguma de parênteses ou de mais variáveis para o cálculo de expressões mais complexas.

4.21. ☼ Elabore um fluxograma que leia um número n (o número de termos de uma progressão aritmética), a_1 (o primeiro termo da progressão) e r (razão) e escreva todos os termos dessa progressão, bem como a soma dos elementos.

4.22. ☼ Construa um fluxograma que leia um número n (o número de termos de uma progressão geométrica), a_1 (o primeiro termo da progressão) e r (razão) e escreva todos os termos dessa progressão, bem como a soma dos elementos.

4.23. ☼ Faça um fluxograma que leia dois valores inteiros e positivos, X e Y. Por meio de multiplicações sucessivas, calcule e exiba a função de exponenciação X^Y.

4.24. ☼ Produza um fluxograma que calcule e exiba o valor da série S a partir de x e n digitados:

$$S = \ln x + x + \frac{x^2}{2} + \frac{x^3}{3} + \ldots + \frac{x^n}{n}$$

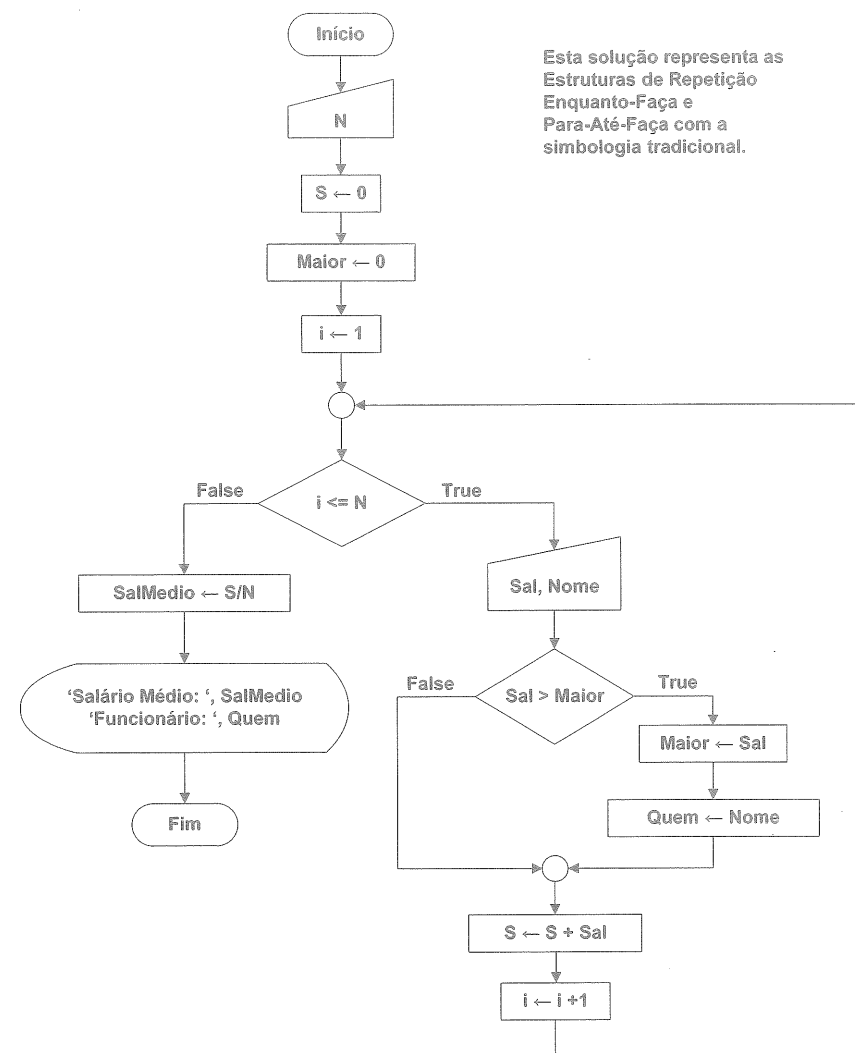
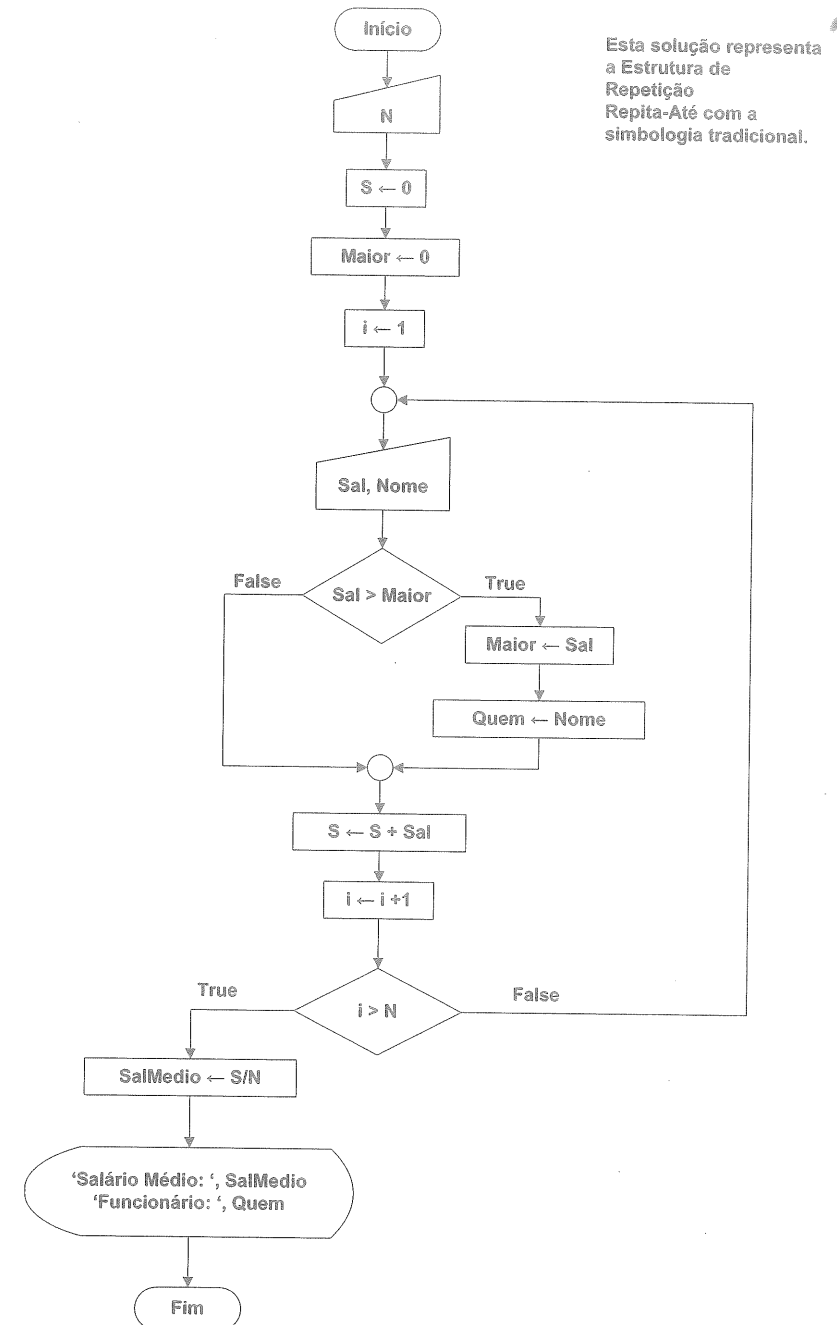
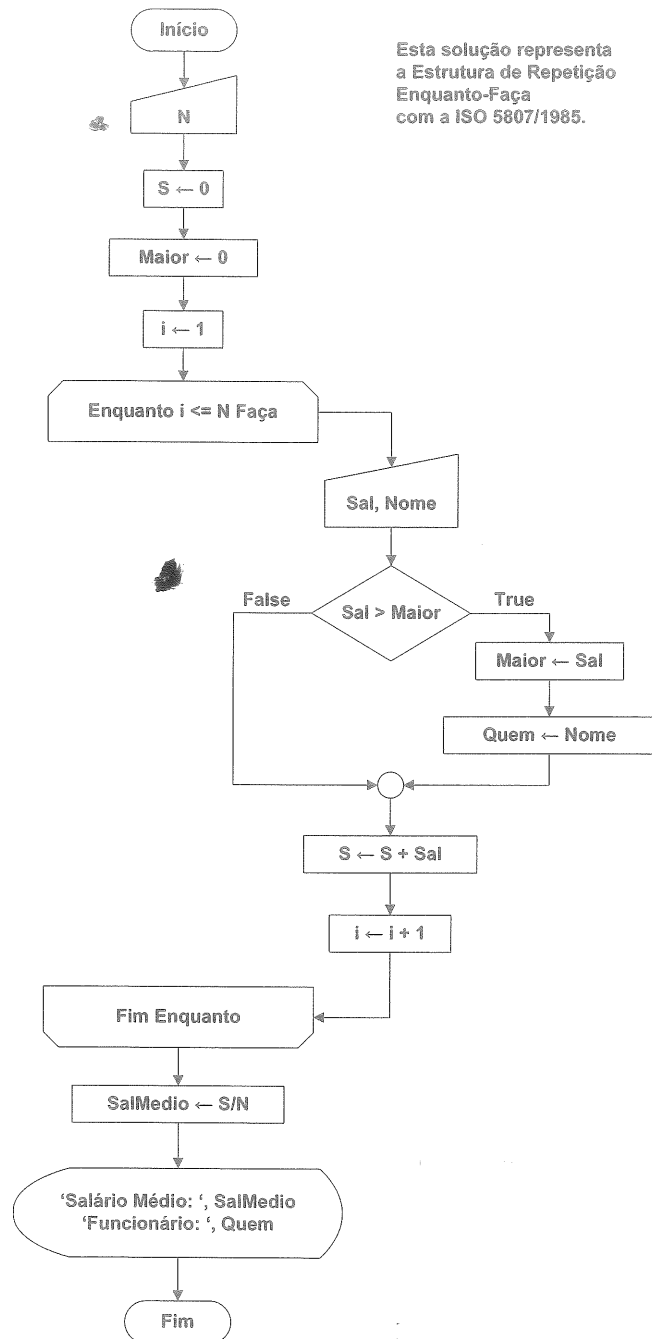
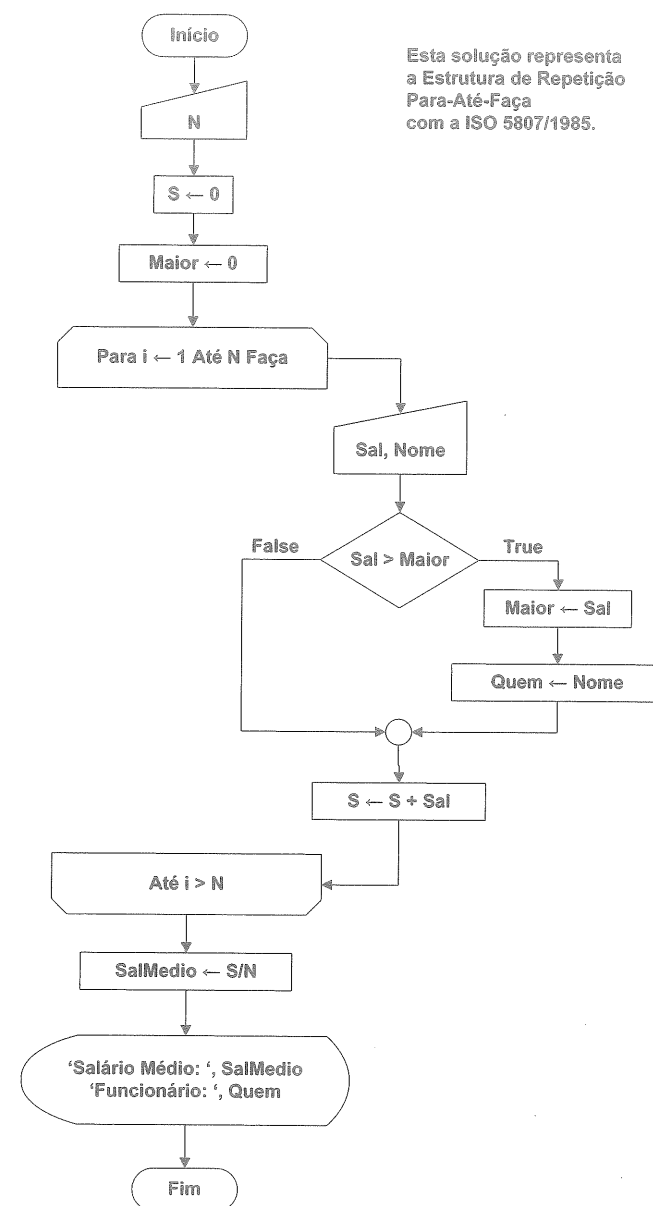
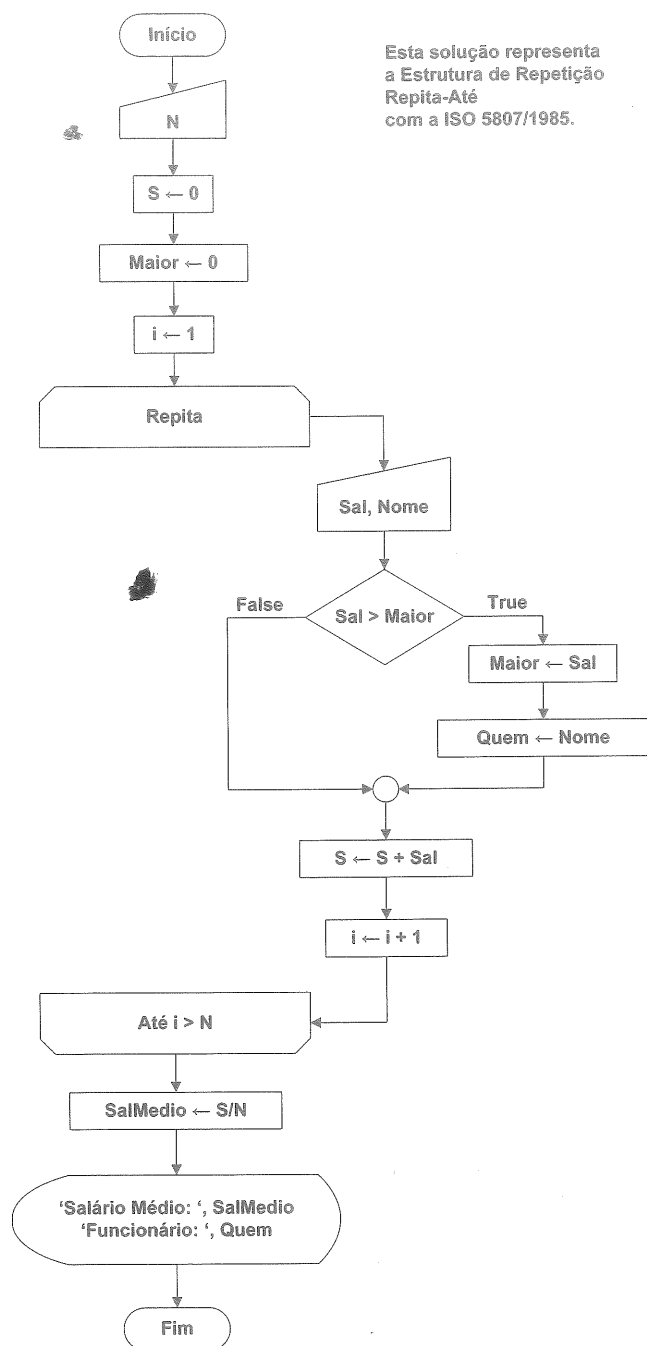
4.7 Exercícios resolvidos

4.5.

Esta solução representa a Estrutura de Repetição Enquanto-Faça com a ISO 5807/1985.

Esta solução representa a Estrutura de Repetição Repita-Até com a simbologia tradicional.

Esta solução representa
a Estrutura de Repetição
Repita-Até
com a ISO 5807/1985.

Esta solução representa
a Estrutura de Repetição
Para-Até-Faça
com a ISO 5807/1985.

Capítulo 5

Variáveis Indexadas

No Capítulo 3 foi apresentado o conceito de variável como uma forma de representar um espaço da memória do computador onde se pode armazenar algum dado. Foi visto também que esse dado possui algum tipo, que se convencionou nas seguintes categorias: números inteiros, números reais, caracteres, cadeias de caracteres e tipos lógicos. Além disso, no Capítulo 4 foi apresentado um conjunto de notações da norma ISO 5807/1985 que permite escrever fluxogramas, representando soluções de problemas envolvendo desde simples conjuntos de comandos sequenciais até comandos que realizam repetições. Embora seja possível somente com os conceitos vistos até agora escrever fluxogramas que possibilitam representar qualquer programa de computador, existem alguns problemas nos quais trabalhar com variáveis simples gera uma limitação na criação e no entendimento de um programa. Neste capítulo será apresentado o conceito de variável indexada, que permitirá a manipulação de grandes massas de dados, proporcionando, assim, a solução de problemas mais interessantes em computação.

5.1 Motivação

Considere o problema de se ordenar, de forma decrescente, três valores inteiros. A ideia é sintetizada pelo Algoritmo 5.1.

Algoritmo 5.1 Algoritmo para ordenar três valores.

Início

1. Ler três valores A, B e C
2. Ordenar os três valores de forma que $A >= B >= C$
3. Exibir os valores A, B e C em ordem.

Fim

Um fluxograma que representa a solução desse problema está ilustrado na Figura 5.1.

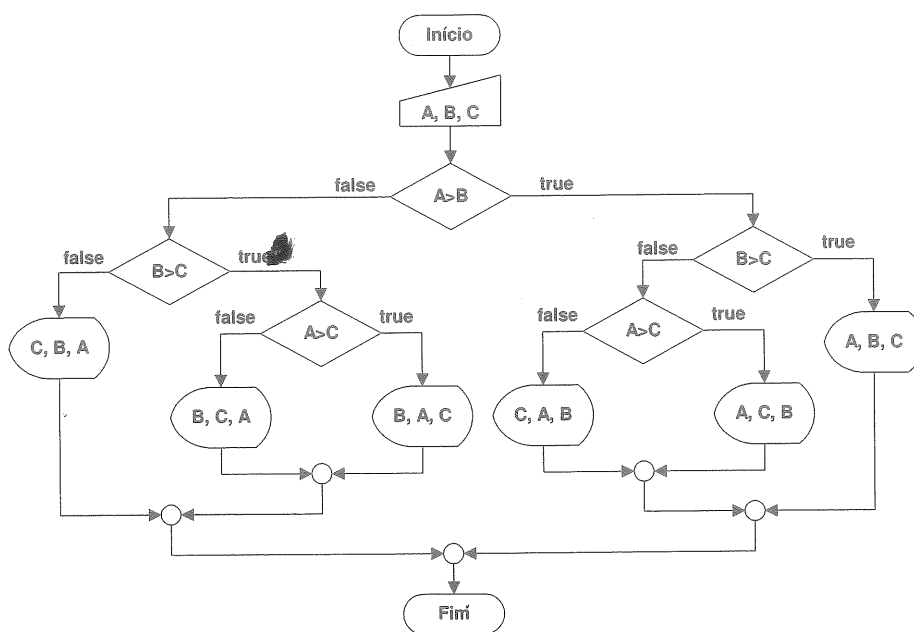



Figura 5.1 Fluxograma para ordenar três valores.

Agora, considerando o mesmo problema, só que para ordenar, de forma decrescente, dez números inteiros. Alguém se arrisca a resolver esse problema conforme o exemplo anterior? Um problema genérico, para ordenar n elementos, vai gerar $n!$ resultados diferentes! Com dez variáveis diferentes, usando o método anterior, deverão ser exibidos 10! (3.628.800) resultados, além de escrever um número de comparações também da ordem de $n!$ (total de losangos a ser desenhados).

Além disso, existe o desconforto de se trabalhar com dez nomes de variáveis diferentes. Assim, precisa-se de uma maneira melhor para representar grandes quantidades

de dados, sem a necessidade de se utilizar nomes de variáveis distintas e sem complicar os algoritmos. Uma forma de resolver esse problema é empregar um tipo especial de variável, que representa um *conjunto ordenado* e *homogêneo de dados*, acessível por um *único nome* e um *índice*. As variáveis desse tipo são denominadas **variáveis indexadas**.

5.2 Variáveis indexadas unidimensionais

As **variáveis indexadas** representam **conjuntos ordenados**[1] de valores homogêneos (isto é, do mesmo tipo), os quais podem ser números inteiros, reais, caracteres, cadeias de caracteres ou ainda valores lógicos.

Tomando como exemplo os prédios de uma grande universidade, como a quantidade de prédios pode ser grande, percebe-se a necessidade de se adicionar algum sistema de localização. Por exemplo, cada prédio poderia ser identificado por uma letra.

Por sua vez, as salas dos prédios podem ser divididas em salas de aula ou escritórios para professores e pessoal administrativo. Da mesma forma que os prédios poderiam ser identificados por letras, as salas de um prédio poderiam ser identificadas por números. Portanto, um prédio poderia ser identificado pela letra Q e suas salas, por números inteiros, como $1, 2, 3, \ldots$.

Dessa maneira, pode-se referir às salas de aula utilizando-se os nomes como $Q1$, $Q2$, $Q3$ etc. Assim, um aluno sabe que a sua aula de Desenho poderia ser, por exemplo, na sala $Q5$, ou seja, a sala 5 do prédio Q.

Então, variando-se apenas o número da sala, é possível determinar o local, isto é, saber de qual sala está se referindo. O número utilizado em questão tem a função de **índice** e, por isso, pode-se afirmar que o bloco Q é indexado. Outro exemplo é a própria lista de presença: o índice é o número de matrícula do aluno, de modo que, localizado o índice, sabe-se onde assinar.

A ideia apresentada nos parágrafos anteriores leva ao conceito de **variável indexada unidimensional**. Uma variável indexada unidimensional é aquela que, a partir de um único nome e de um número (o índice), permite o armazenamento e a localização de um conjunto de dados. As variáveis indexadas unidimensionais também são conhecidas por **arranjos** unidimensionais ou ainda **vetores**, nome, aliás, que será adotado deste ponto em diante neste livro. Os vetores podem ser de qualquer tipo de dado válido, isto é, inteiros, reais, cadeias de caracteres, booleanos etc.

[1]Ordenado no sentido de estarem localizados em alguma posição e não no sentido de estarem respeitando a relação $<$, $\leq$, $>$ ou $\geq$.

5.3 Representação de vetores na memória do computador

À primeira vista, talvez surja um problema: o conceito de variável apresentado até então diz que esta é uma área da memória que pode armazenar um único valor por vez. O conceito de variável indexada discutido na Seção 5.2 mostra uma variável que pode conter diversos valores de uma única vez. Como isso é possível?

Quando se define um vetor, na realidade, está se requisitando ao sistema operacional do computador para que reserve uma área contínua de memória, a fim de armazenar os valores de um mesmo tipo de dado. Essa área da memória é, na verdade, um conjunto de posições simples e contínuas de memória (ou "caixinhas", se preferir) que são reservadas, ficando uma após a outra. A esse conjunto de "caixinhas" é associado um único nome de variável e com o auxílio de um número conveniente – o índice – localiza-se uma "caixinha" específica, cujo valor se deseja manipular.

Voltando ao caso das variáveis comuns, uma variável denominada *A* que guarda o valor 15 é armazenada, na memória do computador, em alguma "caixa" ou posição da memória, como, por exemplo, na posição 100, conforme ilustrado na Figura 5.2.

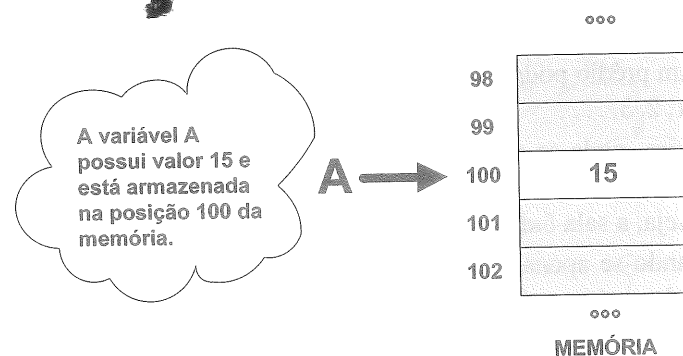



Figura 5.2 Armazenamento de uma variável simples na memória.

Agora, considerando que *A* é um vetor de quatro elementos inteiros, com valores −3, 4, 5 e 0, nesta ordem. Nesse caso, esses valores são armazenados contiguamente em "caixinhas" da memória, a partir de alguma posição inicial, como, por exemplo, 100, conforme mostrado na Figura 5.3.

Observe, neste momento, que agora tem-se um único nome de variável, *A*, e ele representa quatro valores ao mesmo tempo. Cabe, antes de tudo, saber como utilizar essa variável, isto é, como operar com seus valores, mas independentemente do conhecimento de qual posição da memória ela ficará de fato armazenada.

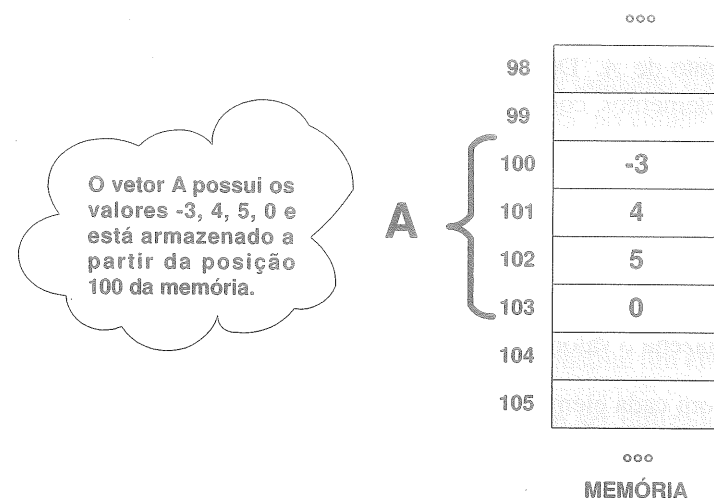



Figura 5.3 Armazenamento de um vetor na memória.

5.4 Utilização de vetores

Embora uma variável tipo vetor armazene um conjunto de elementos simultaneamente, a manipulação desses elementos é individual, como se fossem "um conjunto de variáveis de mesmo nome, identificadas por números individuais".

Os números que identificam os elementos do vetor são denominados índices. Para se localizar algum elemento do vetor, será utilizado o nome da variável vetor seguido dos símbolos "[" e "]", no interior dos quais se especifica o número representando o índice do vetor desejado, conforme apontado na Figura 5.4.

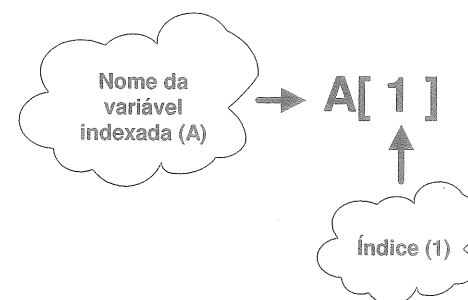



Figura 5.4 Notação para utilizar vetores.

Na Figura 5.4, a expressão $A[1]$ representa a posição da memória que armazena o primeiro elemento de A. Dessa forma, considerando que o vetor A da referida figura possua quatro elementos, como -3, 5, 4 e 0, tem-se:

- $A[1]$ armazena o valor -3;
- $A[2]$ armazena o valor 5;
- $A[3]$ armazena o valor 4;
- $A[4]$ armazena o valor 0.

Opera-se com cada elemento de um vetor realizando as operações descritas no Capítulo 3, como se fossem variáveis independentes, porém ligadas ao nome do vetor. Seguem alguns exemplos:

- $A[1] \leftarrow 6$
- $x \leftarrow A[4]$
- $A[3] \leftarrow 2 * A[3]$
- $A[4] \leftarrow x + A[1]$

As operações executadas são (de acordo com os valores propostos anteriormente):

- No primeiro exemplo, o elemento inicial de A tem seu valor alterado para 6.
- No segundo, uma variável denominada x recebe o valor de $A[4]$. Com os valores propostos, o valor de x será 0.
- No terceiro, o valor de $A[3]$ obtém seu valor atual multiplicado por dois. Como o valor de $A[3]$ é 4, o novo valor de $A[3]$ será 8.
- No último exemplo, o valor de $A[4]$ recebe o valor da variável x somado ao do elemento $A[1]$. Logo, $A[4]$ armazenará o valor -3.

Conclui-se, então, que cada elemento do vetor é operado da mesma forma com a qual se operava com as variáveis convencionais. Para a manipulação de um vetor será preciso utilizar seus elementos individualmente, acessados pelos seus respectivos índices.

Algumas observações **importantes**:

1. Os índices que acessam os elementos de um vetor de tamanho n não precisam ser necessariamente enumerados no intervalo $[1, n]$. Esta, no entanto, é a numeração mais óbvia. Nada impede que se defina um intervalo de índices como $[0, n-1]$ ou $[-3, n-3]$ e assim por diante.
2. Apesar de não ser necessário declarar explicitamente o tamanho do vetor em um fluxograma, ele possui um tamanho máximo definido pelo problema. Utilizar índices que ultrapassem o maior índice do vetor ou que sejam menores que o menor índice de um vetor é uma operação **ilegal** e constitui um **erro** no algoritmo.
3. Somente os números ou as variáveis inteiras podem ser utilizados como índices de um vetor. Por exemplo, se i for uma **variável inteira** contendo um número que está dentro do intervalo de índices de um vetor A, $A[i]$ será uma expressão válida.

5.5 Exemplos de fluxogramas com vetores

5.5.1 Localização de um elemento do vetor

Deseja-se saber o número de pessoas presentes em uma sala específica do bloco Q (contendo seis salas) de uma universidade. Para isso, é necessário um vetor que tenha tamanho 6 e que cada posição armazene o número de pessoas em cada sala. O número de pessoas em cada sala está distribuído de acordo com a Tabela 5.1.

Tabela 5.1 Distribuição de pessoas nas salas de aula.

Sala	Pessoas
1	35
2	4
3	22
4	20
5	36
6	30

Pode-se criar um vetor com seis elementos numerados de 1 a 6, representando uma sala cada um, armazenando em cada posição o número de pessoas presentes na respectiva sala. Será utilizado um vetor denominado Q, cujos elementos serão assim atribuídos:

- $Q[1] \leftarrow 35$
- $Q[2] \leftarrow 4$
- $Q[3] \leftarrow 22$
- $Q[4] \leftarrow 20$
- $Q[5] \leftarrow 36$
- $Q[6] \leftarrow 30$

O algoritmo básico para se resolver esse problema está descrito no Algoritmo 5.2 e seu fluxograma, representado pela Figura 5.5.

Algoritmo 5.2 Algoritmo para armazenar e localizar elementos de um vetor .

Início
1. Armazene os valores nos elementos do vetor.
2. Forneça o número da sala desejada.
3. Exiba a quantidade de alunos na sala desejada.

Fim

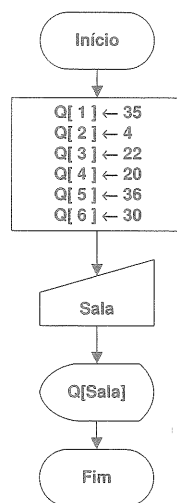



Figura 5.5 Fluxograma para armazenar e localizar elementos de um vetor.

Nesse exemplo, deve-se primeiramente armazenar no vetor Q os valores desejados, correspondentes aos alunos de cada sala. Em seguida, lê-se na variável $Sala$ um número equivalente àquela cuja quantidade de alunos deseja-se consultar.

O bloco de exibição contendo a expressão $Q[Sala]$ mostra o valor existente na posição do vetor Q indicada pelo índice cujo valor está na variável $Sala$. Por exemplo, se ao simular esse fluxograma for digitado o valor 4 na variável $Sala$, o valor a ser exibido será $Q[4]$, ou seja, 20.

No entanto, se for digitado um valor menor que 1 ou maior que 6, haverá a violação dos índices do vetor. Para evitar esse tipo de problema, deve-se acrescentar um teste ao fluxograma, conforme descrito pela Figura 5.6.

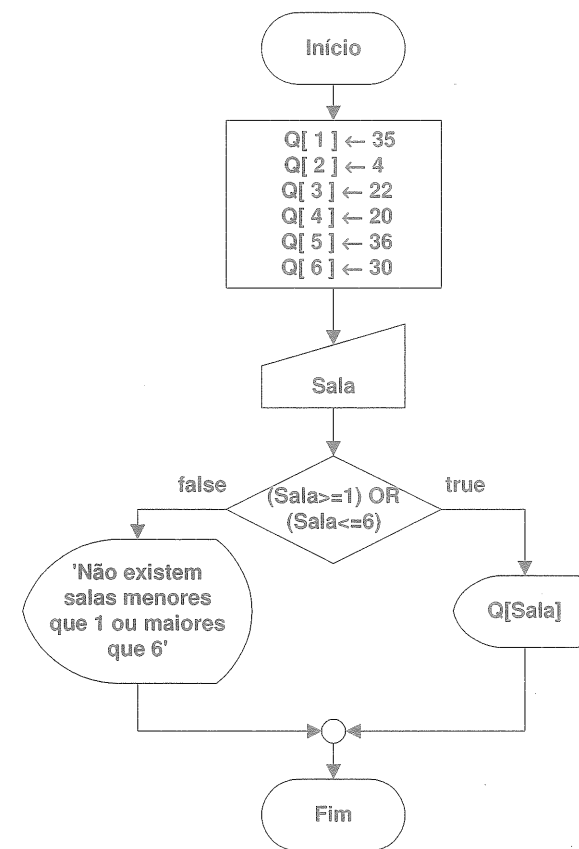



Figura 5.6 Fluxograma seguro para armazenar e localizar elementos de um vetor.

5.5.2 Média aritmética dos elementos de um vetor

Utilizando o mesmo vetor da Seção 5.5.1, será elaborado um fluxograma que permita a entrada do número de alunos de cada sala e, então, exibir a média de alunos por sala. Esse fluxograma está ilustrado na Figura 5.7.

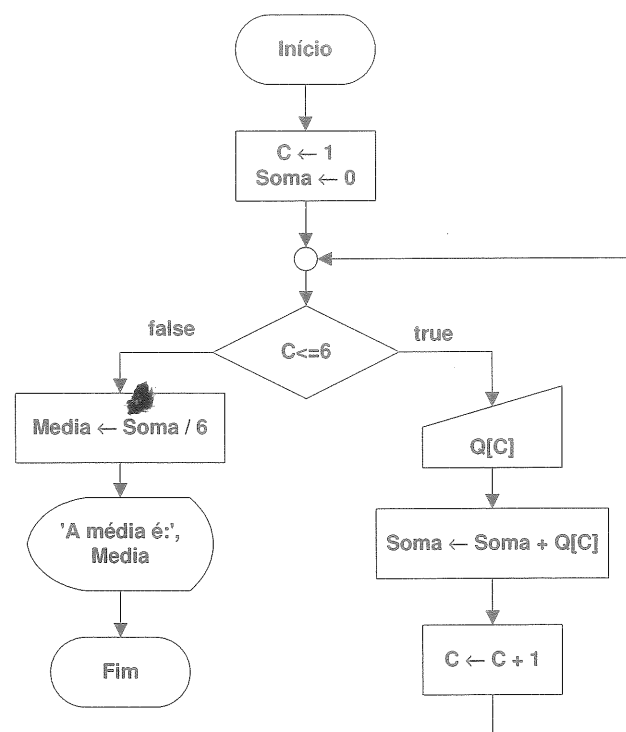



Figura 5.7 Fluxograma para calcular a média aritmética dos elementos de um vetor.

Nesse exemplo, dentro do laço de repetição exibido, é realizada a leitura para o C-ésimo elemento do vetor Q (C é na realidade um contador de 1 a 6). Após a leitura, tem-se o acúmulo do valor digitado com uma soma parcial já existente na variável $Soma$. Por fim, dividindo a $Soma$ pelo número de elementos lidos (6), obtém-se a média.

5.5.3 Localização de elementos de um vetor por algum critério

Na sequência do exemplo da Seção 5.5.2, deseja-se elaborar um fluxograma que leia o número de alunos de cada sala do bloco Q e, então, calcula-se:

- a média do número de alunos por sala;
- para cada soma das salas, calcular quantos alunos acima ou abaixo da média cada uma delas possui.

Nesse caso, basta complementar o exemplo anterior, exibindo a diferença entre a quantidade de alunos em cada sala e a média obtida, como no fluxograma da Figura 5.8.

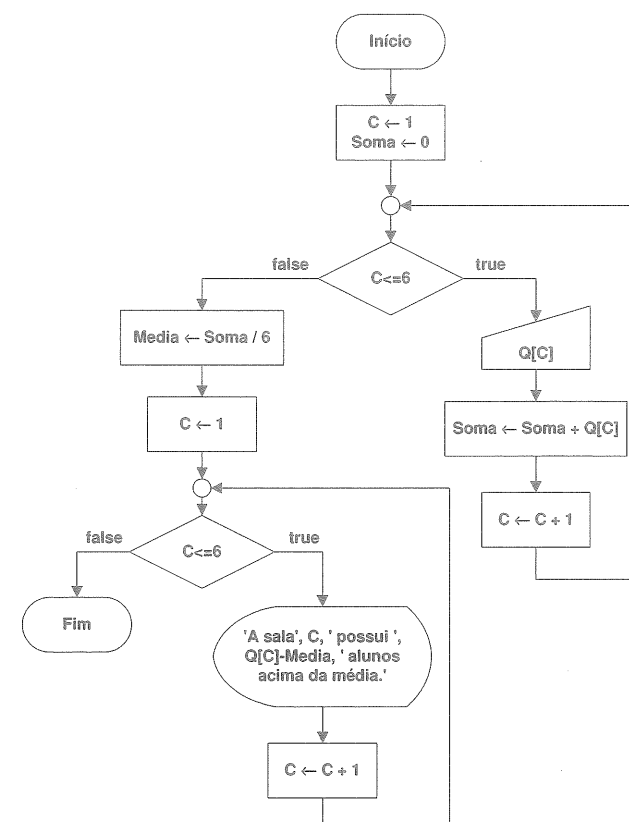



Figura 5.8 Fluxograma para calcular o número de elementos acima e abaixo da média de um vetor.

Observe que é necessário realizar uma repetição para exibir o número da sala (C) e a diferença entre a quantidade de alunos e a média obtida ($Q[C] - Media$).

5.5.4 Determinação do maior e menor elemento de um vetor

Ainda utilizando o mesmo vetor do exercício anterior, deseja-se escrever um fluxograma que permita a entrada do número de alunos de cada sala. Em seguida, deve exibir qual é a sala com o maior número de alunos e qual é esse valor. Repete-se esse mesmo processo para a sala com o menor número de alunos.

Para achar o maior e o menor número dentre os elementos do vetor, procede-se de forma semelhante aos Exercícios 11 e 12 do Capítulo 3, conforme ilustrado na Figura 5.9.

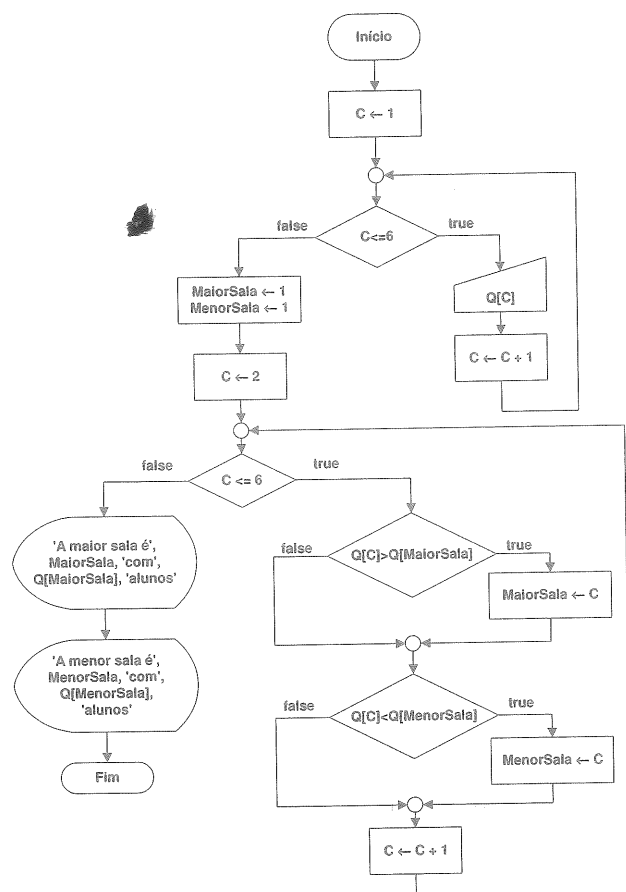



Figura 5.9 Fluxograma para calcular o maior e o menor elemento de um vetor.

5.5.5 Cálculo de um polinômio pelo método de Horner

Como exemplo final desta seção, deseja-se calcular o valor de um polinômio de grau n em um ponto x qualquer. Um polinômio de grau $n \geq 0$ é definido por:

$$P(x) = a_n x^n + a_{n-1}x^{n-1} + \ldots + a_1 x + a_0$$

Uma forma conveniente de calcular o valor em um ponto x qualquer de um polinômio de grau n, $n \geq 0$, é utilizar a regra de *Horner*, conforme descrito a seguir:

- Para $n = 0$: $P(x) = a_0$;
- Para $n = 1$: $P(x) = a_1 x + a_0$;
- Para $n = 2$: $P(x) = (a_2 x + a_1)x + a_0$;
- Para $n = 3$: $P(x) = ((a_3 x + a_2)x + a_1)x + a_0$;
- Para um grau $n > 2$: estender a aplicação das regras anteriores.

Percebe-se que é possível calcular o valor de um ponto x qualquer de um polinômio utilizando apenas multiplicações, sem a necessidade de se calcular explicitamente as potências n-ésimas de cada termo.

Pode-se abstrair esse polinômio para um fluxograma, armazenando seus coeficientes em um vetor A de tamanho $N + 1$. Considerando a variável temporária t com valor inicial igual a $A[N]$, se N for maior ou igual a 1, executam-se N vezes as seguintes instruções, com o valor de i inicialmente igual a N e com valor final igual a 1:

$t \leftarrow t * x$ {*calcular* $a_i * x$}
$t \leftarrow t + A[i-1]$ {*calcular* $a_i * x + a_{i-1}$}
$i \leftarrow i - 1$ {*considerar o próximo termo*}

No final, P (o valor desejado) é igual a t. Se n for igual a 0, o resultado é automático: $P = A[0]$. O fluxograma que resolve esse problema está descrito na Figura 5.10.

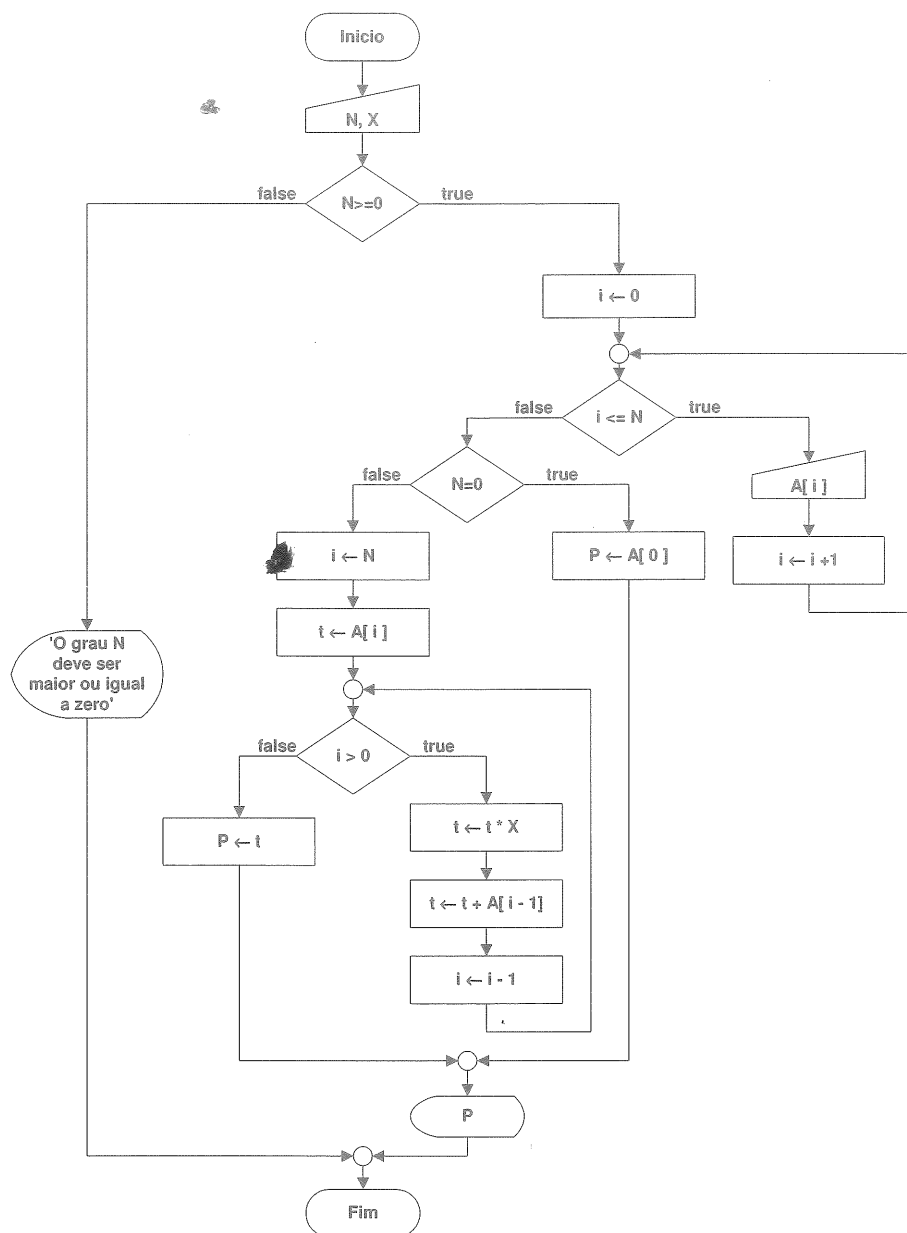



Figura 5.10 Fluxograma para calcular um polinômio pelo método de Horner.

5.6 Variáveis indexadas bidimensionais

Uma matriz bidimensional é uma tabela de valores colocados em linhas e colunas. Nesse caso, para se identificar um entre os diversos valores, é necessário saber em qual linha e em qual coluna ele está armazenado (daí, bidimensional – duas dimensões). As variáveis indexadas bidimensionais são também conhecidas como **tabelas**, **arranjos bidimensionais** ou simplesmente como **matrizes**.

Utilizando novamente a metáfora dos prédios de uma universidade, em prédios que possuem mais de um andar, a localização das salas poderia ser feita com o auxílio de dois números. Por exemplo, em um prédio identificado pelo símbolo P, com dois andares, a localização das salas poderia ser feita da seguinte forma: o primeiro número representa o andar (de baixo para cima) e o segundo número, das salas de cada andar. A distribuição das salas poderia assumir uma configuração segundo a Tabela 5.2.

Tabela 5.2 Distribuição das salas de aula em um prédio com dois andares.

P21	P22	P23	P24
P11	P12	P13	P14

Nesse sistema, a sala $P23$ está no segundo andar, sendo a terceira da direita para a esquerda. Essa tabela poderia servir para indicar a ocupação das salas de aula, preenchendo cada célula com o número de alunos que está em cada sala, conforme a Tabela 5.3.

Tabela 5.3 Tabela de ocupação das salas de aula.

80	65	0	13
95	58	52	120

Esse novo arranjo poderia ser convenientemente simbolizado por $P_{i,j}$, com i variando de 1 a 2 e j variando de 1 a 4. Na notação de elemento de matriz a ser adotado neste livro, cada sala seria representada por $P[i, j]$, em que P é o nome dado à **variável indexada bidimensional** e i e j, os nomes das variáveis inteiras que controlam os índices de linha e coluna respectivamente.

As matrizes com duas dimensões podem ser tratadas da mesma forma que os vetores, observando-se o seguinte (pode ser estendido para mais dimensões):

- Um elemento da matriz será localizado por dois índices. Por exemplo, uma variável indexada bidimensional, denominada $tabela$, tem seu quinto elemento da primeira linha obtido por $tabela[1, 5]$. Assim, refere-se ao 5º elemento da 1ª linha

da tabela. Os valores entre "[" e "]" indicam a posição do elemento e são chamados **índices**: o **primeiro** deles correspondendo à **linha** e o **segundo**, à **coluna**.

- Da mesma forma que em vetores, é possível acessar os elementos de uma matriz utilizando uma forma indireta. Por exemplo, $tabela[n, k]$ representará o 5º elemento da 1ª linha da variável $tabela$, se n for igual a 1 e k igual a 5.
- Ao variar os índices, será possível percorrer os elementos da tabela de diversas formas, dependendo do problema em questão.
- Em qualquer expressão que for utilizado um elemento de uma tabela, será empregado o valor armazenado na tabela e não os índices.
- O comando de atribuição $A[i, j] \leftarrow 6$ será entendido como "armazene o valor 6 na variável A, na posição dada pela linha i e coluna j".
- O comando de atribuição $C[i, j] \leftarrow A[i, j] + B[i, j]$ será compreendido como "recupere o valor que está na linha i e coluna j da tabela A, some com o valor que está na linha i e coluna j da tabela B e armazene o resultado da soma na posição dada pela linha i e coluna j da tabela C". Os valores de $A[i, j]$ e $B[i, j]$ não mudam, somente será alterado o valor de $C[i, j]$.
- O teste $A[i, j] >= A[i, k]$ fornece um resultado verdadeiro ($true$), se o valor da linha i e coluna j da tabela A for maior ou igual ao valor da linha i, coluna k dessa mesma tabela. Isso definirá o caminho a seguir.

5.7 Exemplos de fluxogramas com matrizes

5.7.1 Leitura de elementos para uma matriz

O fluxograma da Figura 5.11 executa a leitura de uma matriz de N linhas e M colunas (denominada A) para a memória do computador (nota-se o uso dos símbolos específicos de repetição da ISO).

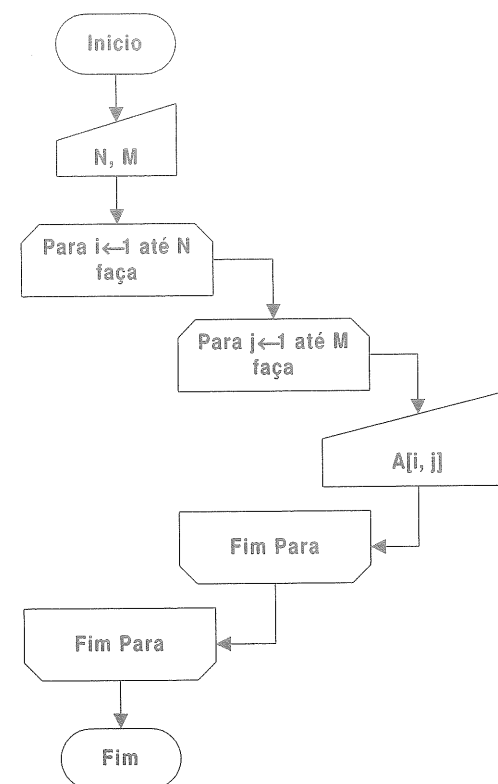



Figura 5.11 Fluxograma para realizar a leitura de uma matriz.

5.7.2 Produto de um vetor por uma matriz

O fluxograma da Figura 5.12 executa o produto de um vetor VET (matematicamente, uma matriz linha), de dimensão N com uma matriz MAT de dimensões $N \times M$ e armazena esse produto no vetor RES (dimensão M). Tanto o vetor como a matriz são lidos inicialmente (em decorrência das dimensões do fluxograma, foi dividido em duas partes, sendo conectado pela "bolinha" – conector – numerada).

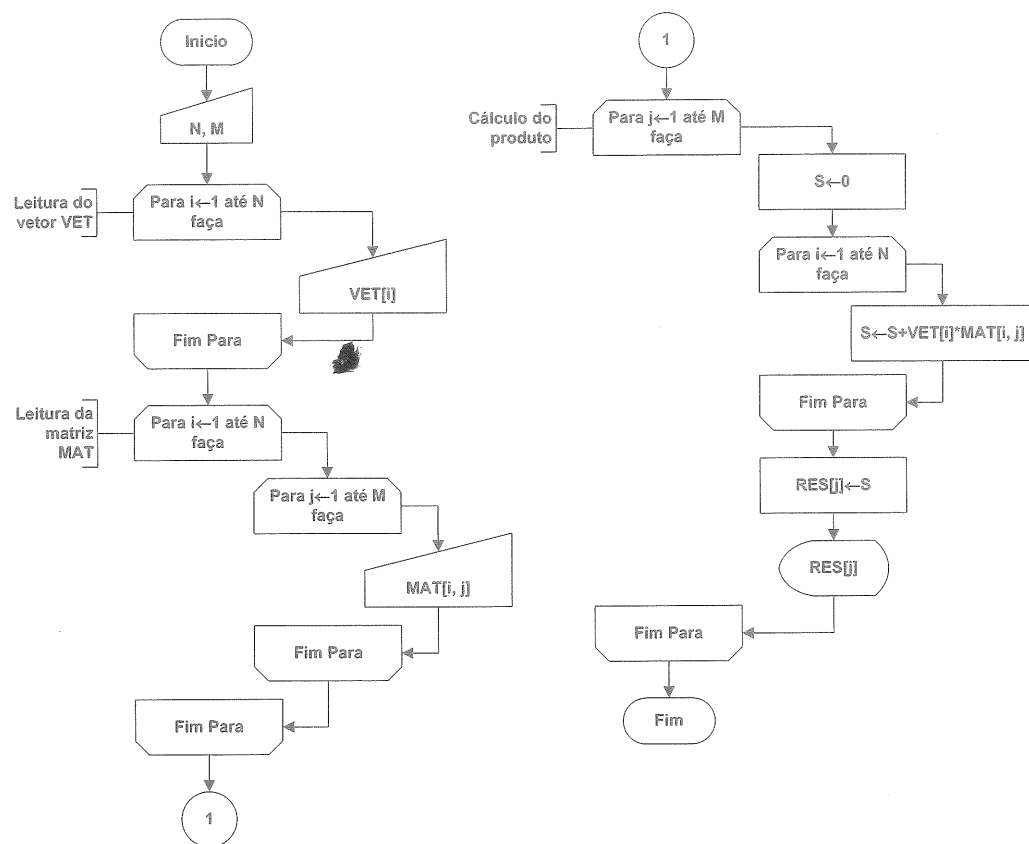



Figura 5.12 Fluxograma para multiplicar um vetor por uma matriz.

5.8 Exercícios

5.1. ☼ Simule os exemplos das Seções 5.5.2, 5.5.3 e 5.5.4 para os seguintes conjuntos de valores a serem armazenados no vetor Q:

- $(33, 35, 32, 40, 25, 20)$;
- $(31, 30, 25, 30, 10, 24)$.

5.2. ☼ Considere dois vetores A e B com cinco elementos indexados a partir de 1. Qual será o valor da variável C a ser exibido pelo fluxograma da Figura 5.13, se forem digitados os seguintes valores para os vetores A e B:

- elementos de A: $(4, 6, 7, 1, 0)$ (nesta ordem);
- elementos de B: $(7, 1, 3, 1, 2)$ (nesta ordem).

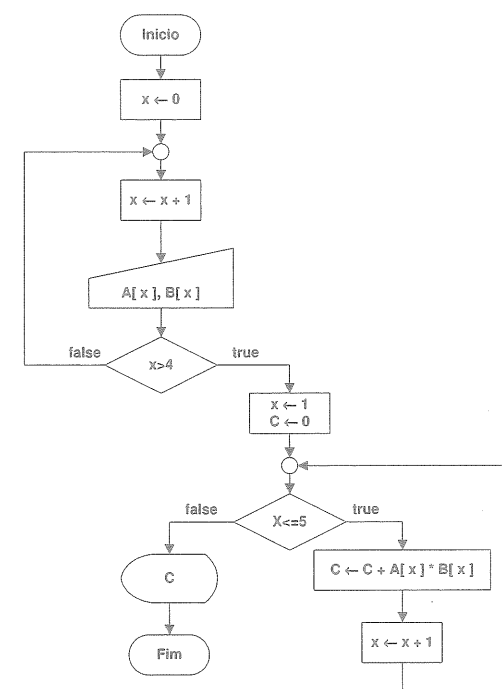



Figura 5.13 Fluxograma para o Exercício 5.2.

5.3. Elabore um fluxograma que calcule e exiba a diferença entre o maior e o menor elemento de um vetor denominado $VALORES$ (com N elementos). Tanto o número de elementos quanto o conteúdo do vetor são valores lidos.

5.4. Construa um fluxograma que leia dois números inteiros a e b, um vetor inteiro de tamanho n e exiba como resposta a contagem de quantos elementos do vetor estão no intervalo fechado $[a, b]$.

5.5. Crie um fluxograma que calcule e exiba o desvio médio (DM) de um vetor x com n elementos. Tanto o número de elementos quanto o conteúdo do vetor são valores lidos.

Fórmulas:

$$\overline{x} = \frac{\sum_{i=1}^{n} x_i}{n}$$

$$DM = \frac{\sum_{i=1}^{n} |x_i - \overline{x}|}{n}$$

5.6. Produza um fluxograma que calcule e exiba o desvio-padrão (σ) de um vetor x com n elementos. Tanto o número de elementos quanto o conteúdo do vetor são valores lidos.

Fórmulas:

$$\overline{x} = \frac{\sum_{i=1}^{n} x_i}{n}$$

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n} |x_i - \overline{x}|^2}{n-1}}$$

5.7. Faça um fluxograma para calcular a maior diferença entre dois elementos consecutivos de um vetor chamado A, com N elementos. Deve-se ler o tamanho do vetor (N) e seus elementos ($A[i]$) antes de mais nada.

5.8. Constitua um fluxograma para efetuar a soma de todos os elementos de índice par de um vetor de tamanho N.

5.9. Forme um fluxograma que leia dois vetores A e B de tamanho N e então troque seus elementos, de forma que o vetor A ficará com os elementos do vetor B e vice-versa.

5.10. Adapte o fluxograma da Figura 5.10 para calcular e exibir o valor desse polinômio para cada x no intervalo $-10 \leq x \leq 10$, variando-se o valor de x de $0,5$ em $0,5$.

5.11. Prepare um fluxograma que calcule o produto escalar de dois vetores da Geometria Analítica. Usar os vetores de tamanho 3 para o fluxograma.

5.12. Elabore um fluxograma que calcule o produto vetorial de dois vetores da Geometria Analítica. Utilizar vetores de tamanho 3 para o fluxograma.

5.13. Crie um fluxograma que calcule o produto misto de três vetores da Geometria Analítica. Utilizar os vetores de tamanho 3 para o fluxograma.

5.14. Realize um fluxograma que calcule a soma de todos os elementos de índice par de um vetor de tamanho N.

5.15. Componha um fluxograma que faça um deslocamento à esquerda de tamanho m vezes (m lido via teclado) em um vetor v de inteiros de tamanho n (n lido via teclado). Por exemplo, a Figura 5.14 apresenta um vetor de tamanho 5, no qual se realiza um deslocamento de tamanho 3.

Figura 5.14 Deslocamento em um vetor.

5.16. Deseja-se construir um sistema de avaliações eletrônico. Para tanto, foi definido que as provas a serem realizadas serão do tipo teste com múltipla escolha (alternativas representadas pelos caracteres 'a', 'b', 'c' e 'd') e que cada prova conterá dez testes. Dessa forma, elabore um fluxograma que permitirá a digitação de um gabarito de uma

prova em um vetor de tamanho 10 e depois a digitação de n valores de nomes de alunos e de suas respostas (utilize mais 2 vetores). Como saída, o fluxograma deverá produzir os nomes e as notas obtidas (de 0 a 10) em uma prova.

5.17. Deseja-se elaborar um fluxograma para representar um programa que verifique se o usuário acertou ou não na Megasena. O fluxograma deverá permitir a entrada de um vetor de tamanho 60 do tipo lógico ($true$ ou $false$) e então ser comparado com outro vetor de tamanho 60 do mesmo tipo. Esses vetores deverão armazenar o valor $true$ nas posições em que o número foi sorteado ou que o usuário apostou. Como resultado, deverá exibir uma mensagem elucidativa, indicando se:

- o usuário não ganhou nada (< 3 acertos);
- o usuário fez uma quadra (4 acertos);
- o usuário fez uma quina (5 acertos);
- o usuário fez uma sena (6 acertos).

5.18. Elabore um fluxograma que percorra um vetor inteiro de tamanho n, realizando trocas 2 a 2, no sentido ascendente, em seus elementos. Por exemplo, a Figura 5.15 apresenta um vetor de tamanho 5, no qual se realizam as trocas entre seus elementos.

Figura 5.15 Troca de elementos em um vetor.

Pergunta-se: como a solução desse exercício poderia auxiliar na resolução do Exercício 15?

5.19. Prepare um fluxograma que permita a entrada de um número n (inteiro) e, então, o converta ao sistema binário, armazenando o resultado em um vetor de tamanho máximo 16. O fluxograma deverá testar, primeiro, se o número não ultrapassa 32.768, que é o maior inteiro que se pode representar com 16 bits (o tamanho do vetor). Depois, o conteúdo desse vetor deve ser exibido na tela.

5.20. Faça um fluxograma para ler duas matrizes, uma $M \times N$ e outra $N \times M$.

5.21. Produza um fluxograma para efetuar a soma dos elementos de uma coluna de uma matriz quadrada de ordem N. O índice da coluna a ser somado deverá ser lido também.

5.22. Forme um fluxograma para efetuar a soma dos elementos de uma matriz quadrada de ordem N.

5.23. Construa um fluxograma para efetuar o produto de um valor por todos os elementos da diagonal principal de uma matriz quadrada de ordem M.

5.24. Elabore um fluxograma para efetuar o produto das duas matrizes, uma $M \times N$ e outra $N \times P$.

5.25. Crie um fluxograma para efetuar o produto da matriz A ($M \times N$) por um vetor coluna V ($N \times 1$).

5.26. Realize um fluxograma para efetuar a soma de todos os elementos de uma matriz quadrada de ordem N, abaixo da diagonal principal.

5.27. Produza um fluxograma para efetuar a soma de todos os elementos de uma matriz de ordem $M \times N$, cuja soma dos índices das linhas e colunas seja par.

5.28. Construa um fluxograma para exibir todos os elementos de uma matriz 8×8, de acordo com o trajeto indicado na Figura 5.16.

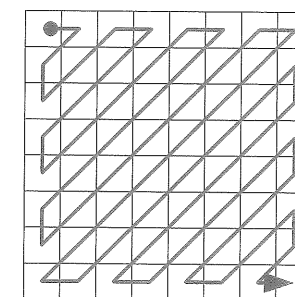



Figura 5.16 Trajeto em uma matriz.

5.29. Um dos algoritmos mais fáceis para se compactar uma imagem é aquele conhecido por RLE (*Run Lenght Encoding*). Basicamente esse algoritmo percorre uma

imagem "bitmap" linha a linha e, se em uma linha ele encontrar dois ou mais bits consecutivos com a mesma cor, ele compacta essa informação escrevendo o número de vezes que o bit foi repetido e em seguida a informação da cor.

Por exemplo, considere que temos uma imagem de 8 por 8 bits, em preto e branco. Uma maneira de representá-la seria por meio de uma matriz 8 × 8, como na Figura 5.17.

0	0	0	0	0	0	0	0
0	0	1	1	1	1	0	0
0	1	0	0	0	0	1	0
1	0	1	0	0	1	0	1
1	0	0	0	0	0	0	1
0	1	0	1	1	0	1	0
0	1	0	0	0	0	1	0
0	0	1	1	1	1	0	0

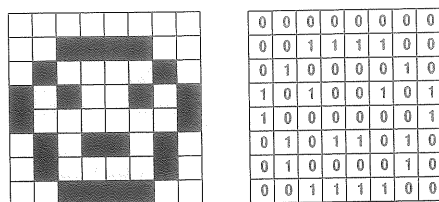



Figura 5.17 Imagem em bitmap.

Os dois únicos valores presentes na matriz são os números 0 e 1, representando, respectivamente, pixels brancos e pretos da imagem. Observe que existe uma repetição de valores 1 e 0 consecutivos em algumas linhas (ou colunas) da matriz.

Pode-se também representar essa imagem como um vetor de caracteres, no qual o valor 0 será representado pela letra 'B' e o valor 1, representado pela letra 'P'. Dessa forma, levando em conta as repetições de cores, pode-se codificar a primeira linha assim (veja a Figura 5.17):

'8 B' (8 pixels repetidos da cor branca)

E a terceira linha desse modo:

'1 B 1 P 4 B 1 P 1 B' (1 branco, 1 preto, 4 brancos, 1 preto e 1 branco)

Agora surge a dúvida: como armazenar a imagem inteira? Pode-se definir o número 0 como indicador de separação de linhas. E o fim da imagem? Utilize dois zeros (00) como marcador de fim da imagem.

Então, a imagem da Figura 5.17 pode ser codificada como se fosse o seguinte vetor de caracteres:

'8B02B4P2B01B1P4B1P1B01P1B1P2B1P1B1P01P6

B1P01B1P1B2P1B1P1B01B1P4B1P1B02B4P2B00'

Pede-se: escrever um fluxograma que leia uma imagem bitmap (valores inteiros 0 e 1) em uma matriz 8 × 8 e que compacte essa imagem em um vetor de tamanho máximo 64, conforme o método discutido anteriormente.

5.30. Escreva um fluxograma que permita descompactar uma imagem, de acordo com o enunciado do Exercício 5.29.

Capítulo 6

Técnicas para a Solução de Problemas

Neste capítulo serão consolidadas as técnicas para a solução de problemas que podem ter tamanho arbitrário. Percebeu-se, por alguns exemplos e exercícios propostos nos capítulos anteriores, que a solução em um único fluxograma de problemas com complexidade média tornava árdua a sua representação, bem como seu entendimento. As técnicas apresentadas neste capítulo seguem o lema de "dividir para conquistar", ou seja, "quebrar" uma solução única, complexa e difícil de entender, por um conjunto de soluções menores, inteligíveis, que juntas formam a solução final. Neste capítulo será apresentada a técnica top-down como aquela que irá permitir "dividir para conquistar", bem como a divisão de um sistema em sub-rotinas, permitindo assim modularizar soluções por meio de funções e procedimentos.

6.1 A técnica *top-down*

A técnica *top-down* possui esse nome porque se **analisa** primeiramente um problema como um todo (*top*), identificando a seguir uma primeira divisão deste em um conjunto de subproblemas menores (*down*). O processo é realizado dessa forma até que não mais seja necessária nenhuma subdivisão.

Depois da identificação e solução dos subproblemas menores, percorre-se o caminho inverso na **síntese** da solução global: juntam-se as soluções menores obtidas de maneira ordenada até se formar a solução procurada.

Em síntese, essa técnica pode ser assim descrita:

1. Entender o problema a ser resolvido.
2. Estabelecer o objetivo a ser alcançado.
3. Dividir o problema (solução desconhecida) em problemas menores, com solução mais simples (ou conhecida) e cujo total permita atingir o objetivo.
4. Continuar a subdividir os problemas gerados até que seja possível solucionar a todos.
5. A solução do problema original é feita pela junção ordenada das soluções dos problemas finais.

Como motivação para a utilização dessa técnica, considere a criação de um projeto mecânico. Algumas pessoas poderiam desenvolver o projeto mostrando todos os detalhes em um único desenho. Sabe-se, porém, que isso é simples se o projeto for rudimentar, mas, caso a complexidade aumente, isso nunca funcionará.

O desenho de uma peça para posterior fabricação segue um processo conhecido. O desenho deve ser representado com uso de pelo menos três vistas: a planta, a elevação e a lateral. Qual a razão disso? A razão está em exibir os diferentes detalhes construtivos da peça de uma forma racional, simples e que não deixe margem a dúvidas.

Mesmo assim, existem situações em que surgem dúvidas; nesse caso deve-se efetuar cortes na peça, mostrando os detalhes de forma a torná-los evidentes e dirimir as eventuais dúvidas. Não exibir esses detalhes pode fazer com que o produto final não tenha a função originalmente projetada (a peça não serve para aquilo a que foi especificada) e expor todos os detalhes em um único desenho dificulta demais a visualização.

Na solução de problemas de âmbito computacional, ocorrem situações semelhantes. Com o crescimento do número de comandos a serem utilizados, pode-se facilmente perder a visão do todo, e aquele comando mal posicionado pode pôr a perder todo o trabalho de desenvolvimento (identificar um comando errôneo em um único e grande programa é como achar uma agulha no palheiro).

6.1.1 Exemplo de aplicação

Como exemplo de aplicação da técnica *top-down*, será analisado o seguinte problema: deseja-se construir um sistema automatizado para calcular as notas finais, em todas as disciplinas, de todos os alunos de uma escola. Como isso pode ser feito? A seguir, repetem-se os passos da técnica com foco neste problema.

Entendimento do problema

O problema exige que se tenha as notas de cada aluno por disciplina. Com esses dados, deve-se calcular e exibir as notas finais.

Estabelecer o objetivo a ser alcançado

O objetivo é exibir as notas finais de todos os alunos para todas as disciplinas da escola.

Divisão do problema

Analisando esse problema de maneira informal, apenas para se ter um primeiro contato com a técnica *top-down*, seria possível elaborar o fluxograma da Figura 6.1, representando o ponto de partida em busca da solução do problema das notas.

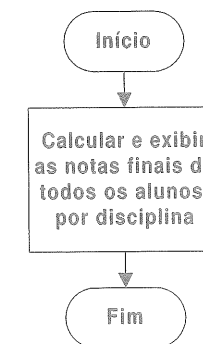



Figura 6.1 Ponto de partida na busca da solução do problema das notas.

Percebe-se que são necessários alguns refinamentos. Dirige-se, então, para o particionamento do problema.

Particionamento do problema

Em uma primeira divisão do problema, descobre-se que é necessário o cálculo das notas dos alunos para cada disciplina, conforme a Figura 6.2.

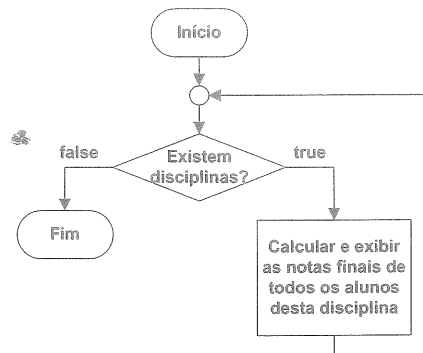



Figura 6.2 Primeira partição do problema.

Continuando a subdividir o problema, o próximo refinamento seria calcular e exibir as notas finais de cada aluno para cada disciplina, conforme ilustrado na Figura 6.3.

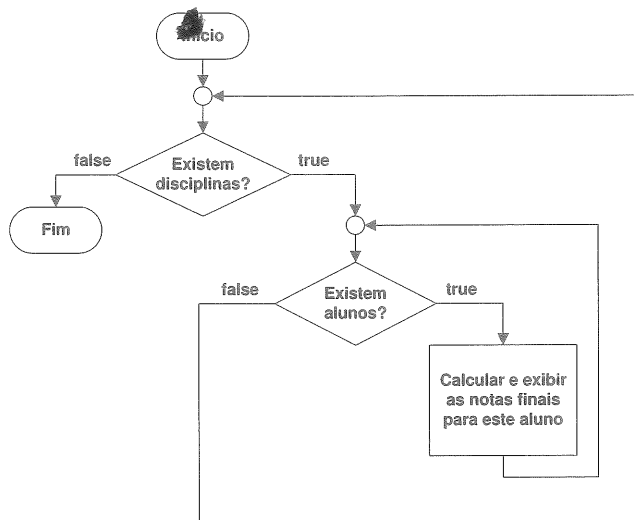



Figura 6.3 Segunda partição do problema.

O próximo refinamento seria definir como calcular e exibir as notas finais para um único aluno, conforme indicado na Figura 6.4. Neste ponto, cabe refinar o processo de cálculo da nota final. Tome como exemplo a expressão apresentada a seguir (supondo que o aluno realize quatro provas no ano):

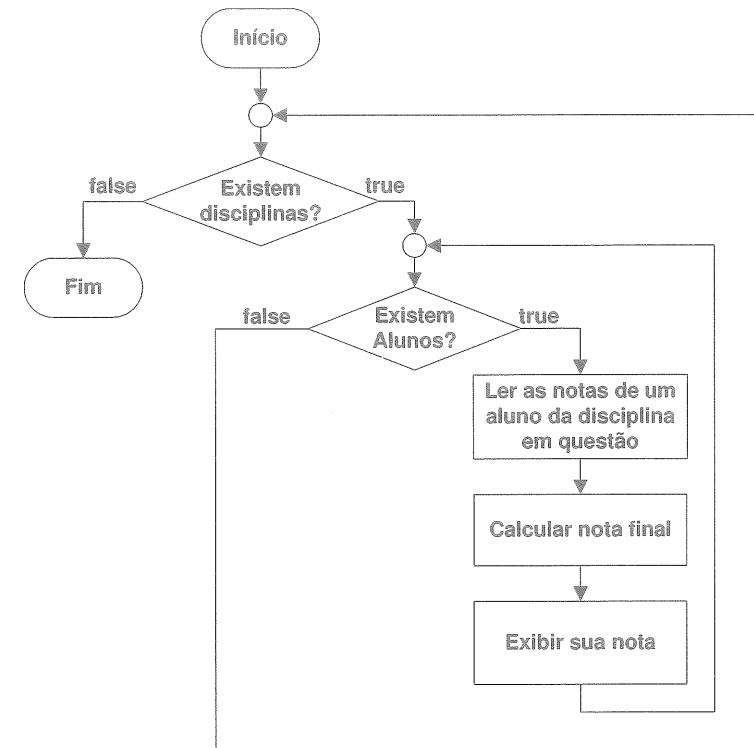



Figura 6.4 Terceira partição do problema.

$$P = \frac{P' + 2P'' + 3P_4}{6}$$

Em que:

- P'' é a maior nota entre P_1, P_2 e P_3;
- P' é a segunda maior nota entre P_1, P_2 e P_3.

Com o conhecimento dessa fórmula, basta agora refinar o processo *Calcular Nota Final* a fim de que primeiro ordene as notas de modo decrescente (para descobrir a maior

e a segunda maior notas) e então aplicar a fórmula. O fluxograma final está apresentado na Figura 6.5.

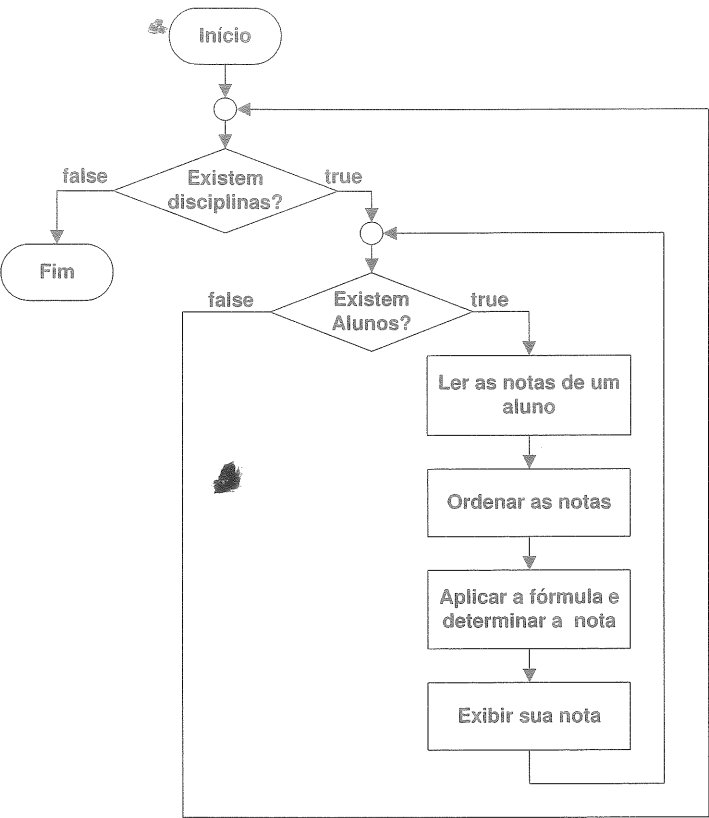



Figura 6.5 Partição final do problema.

Solução do problema original

Percebe-se que as soluções encontradas já foram colocadas em seus lugares. Para uma versão final, ainda é necessário definir com exatidão os processos apresentados informalmente nesse fluxograma. Para se fazer isso, pode-se contar com *sub-rotinas*, assunto das próximas seções.

6.2 Sub-rotinas

Sub-rotina ou ainda subalgoritmo é o nome dado a um algoritmo que realiza uma tarefa específica, mas não representa um algoritmo (ou solução) completo. As sub-rotinas são utilizadas para a resolução de partes (integrantes e distintas) de algoritmos. Seu uso é aconselhável principalmente para os seguintes casos:

- Evitar que o algoritmo torne-se complexo: nesse caso a sub-rotina pode representar partes específicas da solução de um problema maior, tornando mais simples o entendimento da solução como um todo (é uma aliada poderosa da técnica *top-down*).

- Incentivar a reutilização de algoritmos: por exemplo, se fosse escrita uma sub-rotina para ordenar um vetor qualquer de tamanho N, esta poderia ser utilizada em qualquer problema que exigisse a ordenação de um vetor de tamanho predefinido, sem a necessidade de reescrevê-lo.

Existem dois tipos de sub-rotinas, que serão descritas a seguir: funções e procedimentos.

6.2.1 Funções

As funções são sub-rotinas que retornam um valor calculado. O conceito de função torna-se simples de entender se for lembrado o conceito de funções na Matemática, como, por exemplo, a função *seno*.

Na Matemática, ao se escrever $x = sen(0{,}77)$, está se aplicando a função *seno* sobre um argumento que, neste exemplo, é 0,77. Esse resultado é então atribuído à variável x. Não é necessário saber *a priori* como realmente funciona essa função, pois esses detalhes se encontram no algoritmo de cálculo da função seno de uma calculadora e em textos de cálculo numérico. Assim, em todas as situações em que se deseja o seno de um número, acaba-se por reutilizar essa função.

Em fluxogramas, as funções são representadas da seguinte forma (notar o símbolo de início para funções), conforme ilustrado na Figura 6.6:

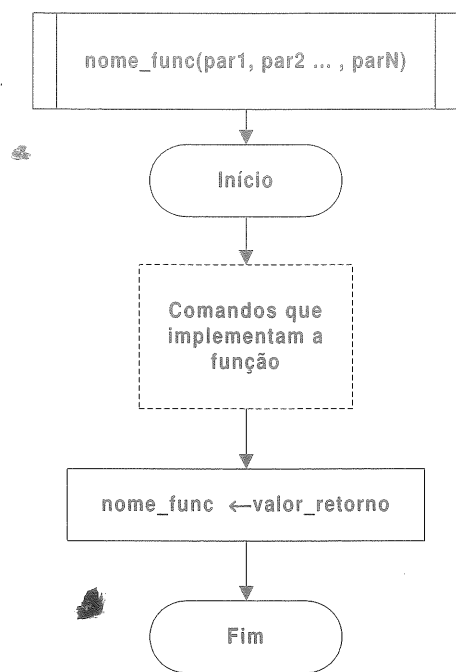



Figura 6.6 Representação de função em fluxograma.

- $nome_func$: representa o nome da função. Utiliza-se qualquer nome para uma função, desde que esteja de acordo com as regras para as variáveis descritas no Capítulo 3.

- $(par1, par2, \ldots, parN)$: representa uma lista de parâmetros sobre os quais a função vai operar. É com esses valores que ela vai calcular um resultado final a ser retornado. Pode-se escrever uma função sem parâmetro algum; nesse caso, não se utilizam parênteses. Cada parâmetro pode ser de qualquer tipo (inteiro, real, lógico, caractere, cadeia de caracteres, vetor ou matriz).

- $nome_func \leftarrow valor_retorno$: essa expressão é obrigatória para funções e indica que esse fluxograma representa uma função. Se a função se chamar $CalcAlgumaCoisa$ e for retornar um valor final que está em uma variável X da função, essa expressão ficaria: $CalcAlgumaCoisa \leftarrow X$. O retorno pode ser de qualquer tipo (inteiro, real, lógico, caractere, cadeia de caracteres, vetor ou matriz).

6.2.2 Exemplos de funções

Deseja-se escrever uma função que calcule a contribuição para o INSS (a contribuição é calculada segundo a Tabela 6.1).

Tabela 6.1 Tabela de contribuições ao INSS.

TABELA VIGENTE **Tabela de contribuição dos segurados empregado, empregado doméstico e trabalhador avulso, para pagamento de remuneração a partir de 16 de junho de 2010 Portaria nº 408, de 17 de agosto de 2010**	
Salário de contribuição (R$)	**Alíquota para fins de recolhimento ao INSS (%)**
até R$ 1.040,22	8,00 %
de R$ 1.040,23 a R$ 1.733,70	9,00 %
de R$ 1.733,71 até R$ 3.467,40	11,00 %
acima de R$ 3.467,40	valor fixo de R$ 381,41

A primeira decisão a ser tomada refere-se aos parâmetros da função. Para esse exemplo, nota-se que o cálculo da contribuição ao INSS é feito de acordo com o salário do contribuinte. Assim, uma função para se calcular a contribuição deve ter um parâmetro: salário (um número real). O retorno da função será um número real que representa o valor da contribuição. Dessa forma, o fluxograma que define essa função é exposto na Figura 6.7.

O identificador S é o parâmetro dessa função. Ao ser utilizada, deverá ser passado algum valor para S de modo que a função retorne um resultado. Observe que, internamente a essa função, utiliza-se a variável C para armazenar de forma temporária o resultado a ser retornado. Essa função poderia ser utilizada para o cálculo do desconto do IRRF (Imposto de Renda Retido na Fonte). Esse cálculo é feito sobre o salário líquido após a dedução da contribuição ao INSS, de acordo com a Tabela 6.2.

Tabela 6.2 Tabela de descontos para o IRRF.

Base de Cálculo em R$	Alíquota %	Parcela a Deduzir do Imposto em R$
Até 1.499,15	–	–
De 1.499,16 até 2.246,75	7,5	112,43
De 2.246,76 até 2.995,70	15	280,94
De 2.995,71 até 3.743,19	22,5	505,62
Acima de 3.743,19	27,5	692,78

Líquido = Bruto-INSS-IR
Líquido = Bruto-INSS-(base*alíquota - parcela)

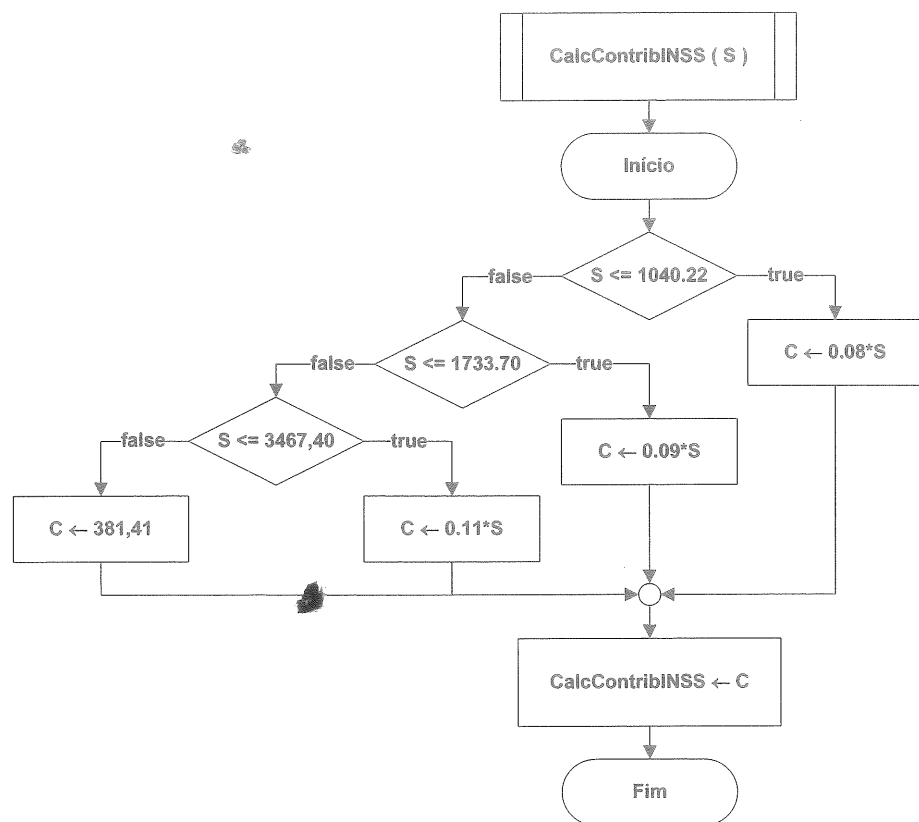



Figura 6.7 Função para calcular a contribuição do INSS.

Utilizando a função $CalcContribINSS$ definida anteriormente, o fluxograma para esse cálculo ficaria conforme a Figura 6.8.

Como o cálculo do IRRF é muito executado em várias situações (folha de pagamento, por exemplo), cria-se também uma função que faça esse cálculo, segundo a Figura 6.9. Nota-se que as funções podem executar outras funções!

Finalmente, a solução para o cálculo de IRRF pode ser assim simplificada, conforme a Figura 6.10.

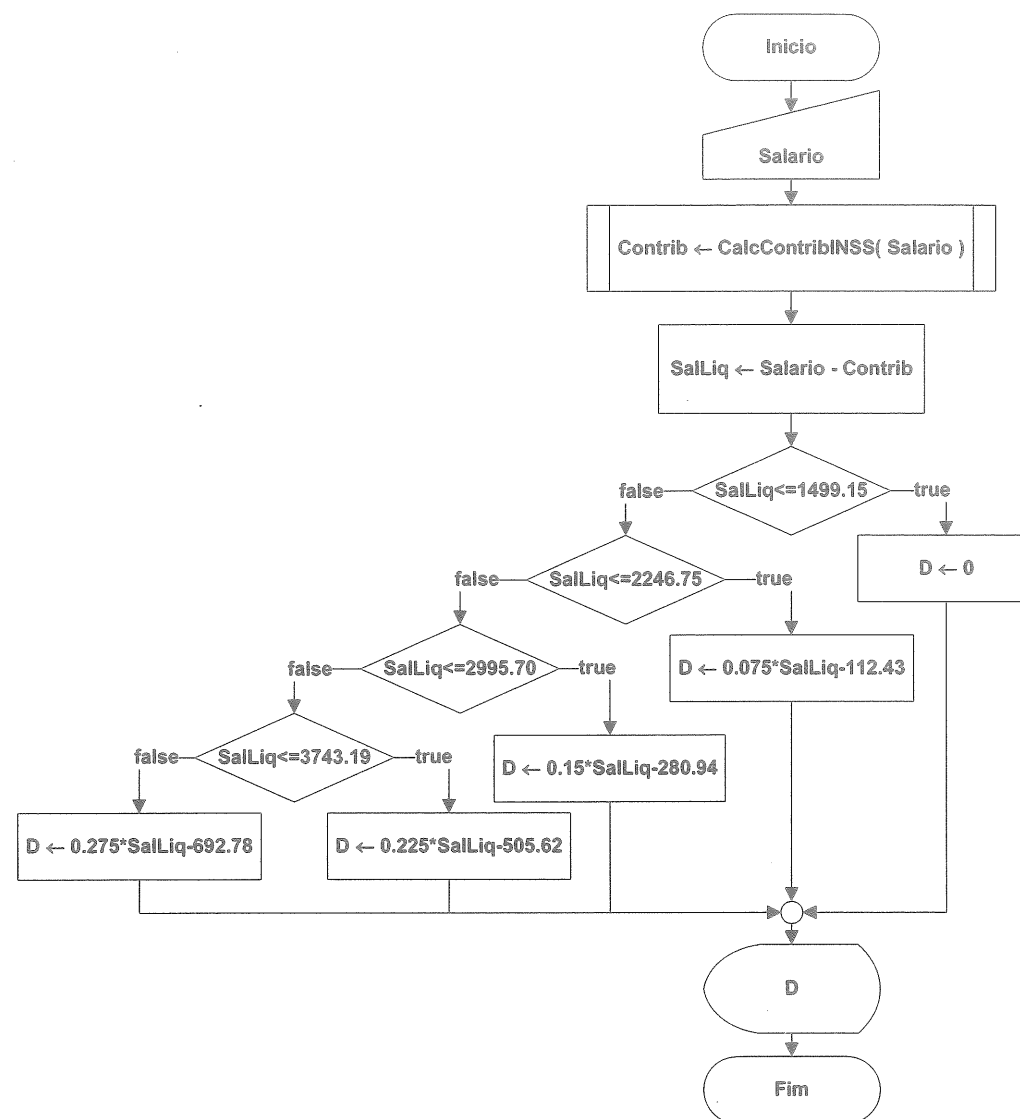



Figura 6.8 Uso da função $CalcContribINSS$.

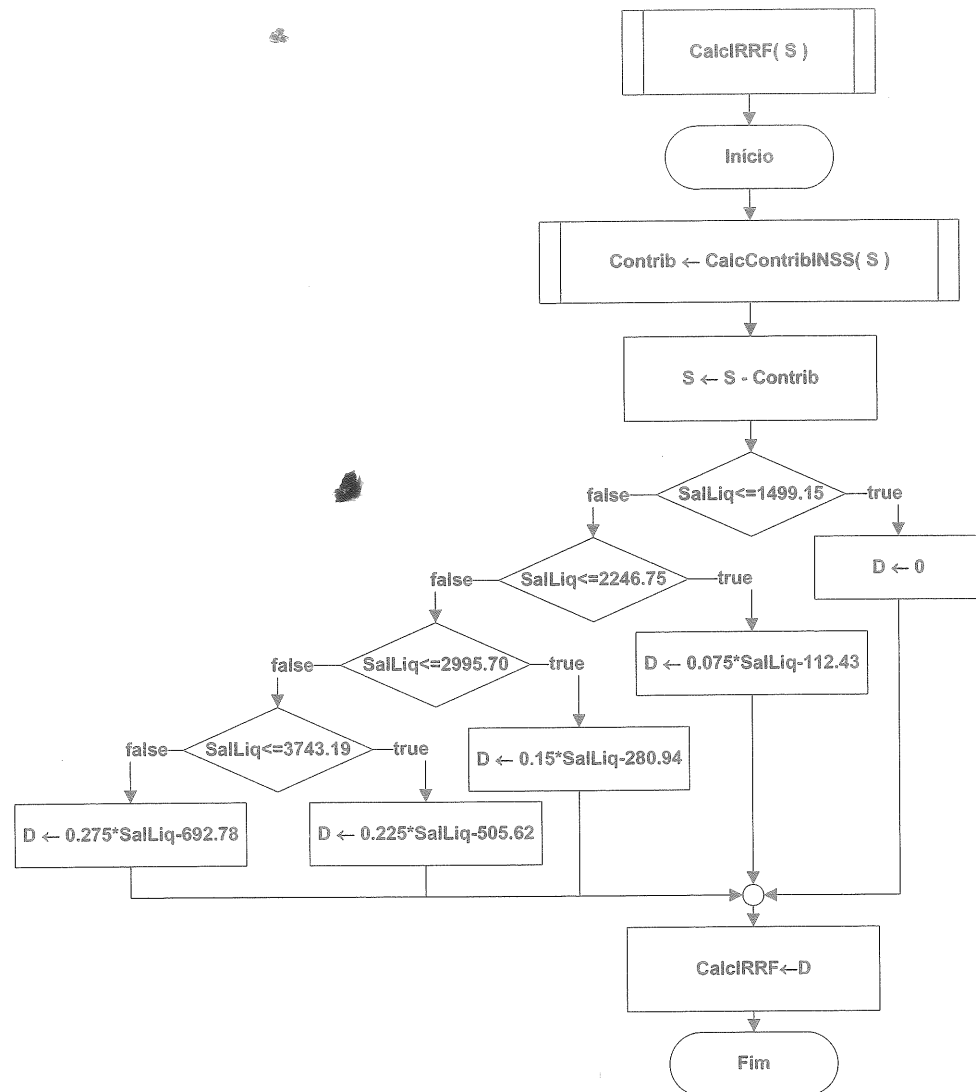



Figura 6.9 Função para calcular o desconto do IRRF.

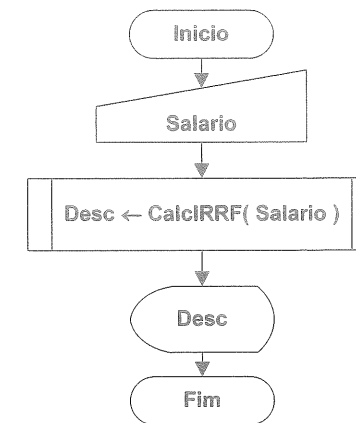



Figura 6.10 Fluxograma simplificado para o cálculo do IRRF.

6.2.3 O mecanismo de chamada de funções

Ao se executar uma função, ocorre um desvio no fluxo para o diagrama no qual a função está definida. Nesse desvio, executa-se a função, obtém-se um valor de retorno e volta-se para o fluxograma original no comando logo após o ponto em que se executou a função. Veja esse mecanismo, tomando como exemplo a execução do último fluxograma apresentado, conforme ilustrado na Figura 6.11.

Observações:

1. Ao se executar uma função, o seu parâmetro recebe implicitamente o valor de uma constante ou variável que se substitui em seu lugar. Quando se escreve uma chamada a uma função como $Desc \leftarrow CalcIRRF(Salario)$ entende-se que se o valor digitado para a variável $Salario$ for 10, ocorrerá implicitamente $S \leftarrow Salario$ e então a função vai executar os cálculos sobre seu parâmetro S, que contêm o valor de $Salario$. A esse tipo de passagem de parâmetro, dá-se o nome de "passagem de parâmetro por valor" ou, simplesmente, "parâmetro por valor" ou, ainda, "passagem por valor";

2. Não há problema algum em se reutilizar nomes de variáveis e parâmetros idênticos em funções diferentes. Reconheça que a função "protege" suas variáveis e parâmetros de outras de mesmo nome existentes em funções externas a ela;

3. Uma função retorna sempre um único valor.

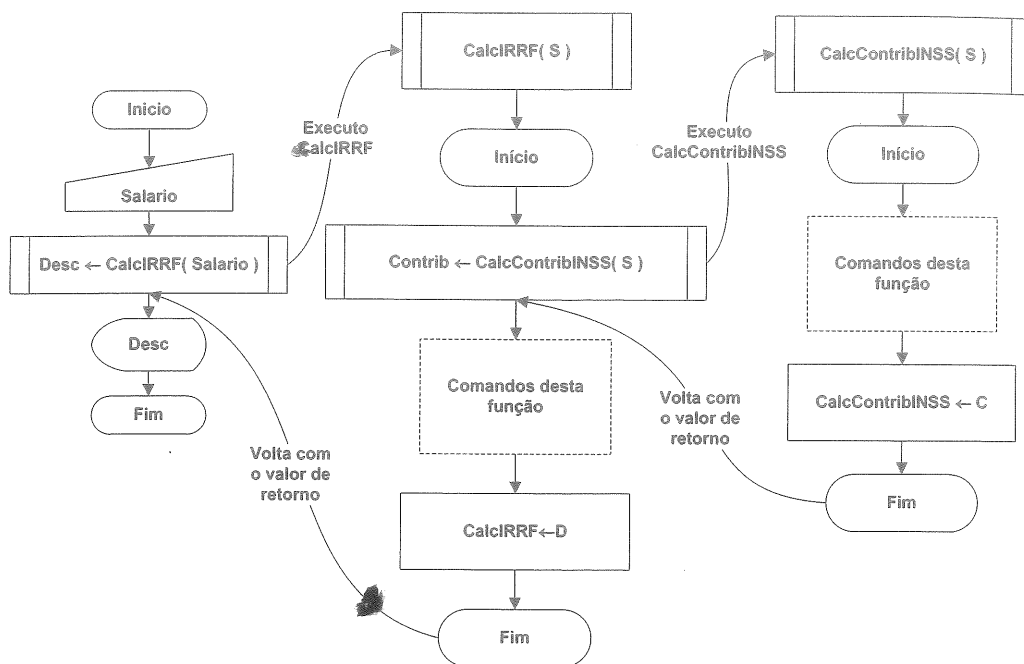



Figura 6.11 O mecanismo de chamada de uma função.

6.2.4 Procedimentos

Os procedimentos são sub-rotinas que, ao contrário de funções, não retornam um valor explicitamente. O uso de procedimentos é aconselhável quando se deseja realizar uma operação na qual uma função não se encaixa muito bem.

Um exemplo de aplicação de procedimento poderia ser no algoritmo que ordene os N valores de um vetor. Nesse caso, a tarefa se resume em ordenar um vetor existente, sem a necessidade de se retornar um valor.

Em fluxogramas, os procedimentos são representados de acordo com a Figura 6.12, sendo:

- *nome_proc*: representa o nome do procedimento. Utiliza-se qualquer nome para um procedimento desde que esteja de acordo com as regras para variáveis vistas no Capítulo 3.

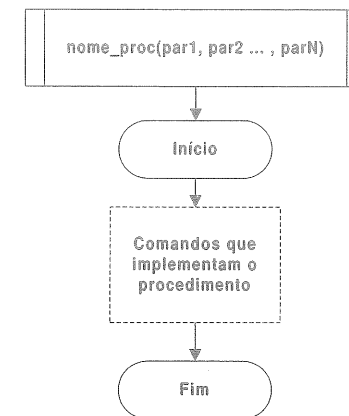



Figura 6.12 Representação de um procedimento.

- $(par1, par2, \ldots, parN)$: representa uma lista de parâmetros que o procedimento vai receber. Pode-se escrever um procedimento sem parâmetro algum; nesse caso, não se utilizam parênteses. Cada parâmetro pode ser de qualquer tipo (inteiro, real, lógico, caractere, cadeia de caracteres, vetor ou matriz).

- Como um procedimento não retorna os valores, não existe o bloco indicando esse retorno.

Como exemplo, tem-se um procedimento para realizar a leitura de N valores de um vetor qualquer, representado pelo parâmetro V, segundo a Figura 6.13.

Esse procedimento poderia ser utilizado em outro fluxograma que necessite ler os valores para um vetor específico com tamanho arbitrário, como ilustrado pela Figura 6.14.

Algumas observações:

1. Ao se executar um procedimento, o seu parâmetro recebe implicitamente o valor de uma constante ou variável que se substitui em seu lugar (como em funções). Esse tipo de parâmetro recebe o nome de parâmetro por valor (veja o item 6.2.3);

2. Não há problema algum em se reutilizar nomes de variáveis e parâmetros idênticos em procedimentos diferentes. Reconheça que o procedimento "protege" suas variáveis e parâmetros de outras de mesmo nome existentes em procedimentos externos a ele. Nesse tipo de passagem por valor, se o conteúdo do parâmetro for alterado por alguma expressão dentro da sub-rotina, ele não será enviado para

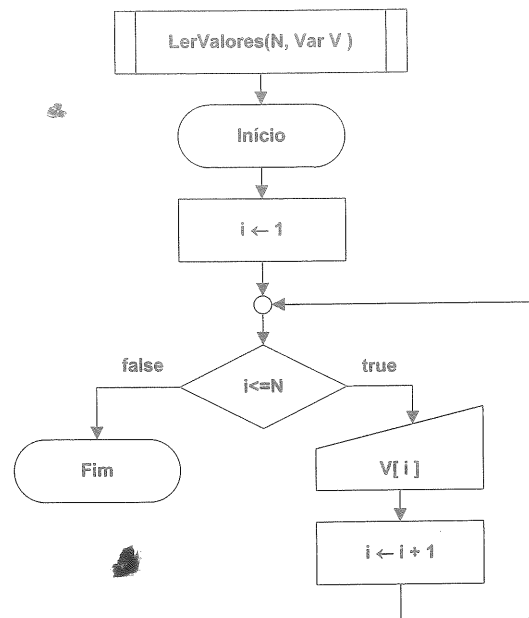



Figura 6.13 Procedimento para realizar a leitura de um vetor.

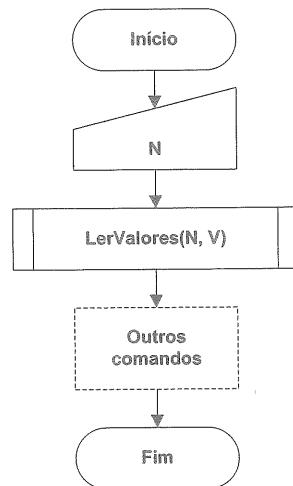



Figura 6.14 Utilização de um procedimento.

quem o chamou. Assim, pode-se dizer que esse tipo de parâmetro tem a característica de ser um "parâmetro de entrada", ou seja, apenas recebendo e nunca enviando o seu conteúdo;

3. Presume-se que os parâmetros tanto de funções quanto de procedimentos possam ser alterados internamente. Veja o fluxograma do exemplo anterior;

4. Na Figura 6.14, o parâmetro *V* sofre a passagem por referência. Nesse tipo de passagem é enviado ao parâmetro o endereço de memória onde a variável foi criada. Por meio dessa característica, se o valor do parâmetro for alterado, também sofrerá alteração a variável referenciada pelo parâmetro. Assim, pode-se dizer que esse parâmetro tem a característica de ser um "parâmetro de entrada e saída", ou seja, recebe e envia o seu conteúdo;

5. Para diferenciar os dois tipos de passagem por parâmetro, adota-se nesse livro a palavra *VAR* no cabeçalho da sub-rotina antes do parâmetro que sofre a passagem por referência (veja a Figura 6.13);

6. Já a chamada da sub-rotina continua sendo feita da maneira usual, ou seja, sem nenhuma alteração (veja a Figura 6.14).

6.3 Exercícios

6.1. Escreva o fluxograma para a função fatorial (utilize os conceitos de sub-rotinas apresentados). Lembrando:

$$n! = n \cdot (n-1) \cdot (n-2) \cdot (n-3) \cdot \ldots \cdot 1$$

6.2. Escreva um fluxograma para calcular a soma dos N primeiros termos da série:

$$S = \frac{1!}{2!} + \frac{3!}{4!} + \frac{5!}{6!} + \ldots$$

Em que N é lido pelo teclado. Utilize a função fatorial escrita anteriormente.

6.3. Deseja-se ler N pares de números e então calcular o produto dos maiores números em cada par. Como se procederia na solução desse problema, empregando a técnica *top-down*? Devem ser definidas as funções e os procedimentos necessários para a solução.

6.4. Utilizando a técnica *top-down*, como poderia ser implementada uma calculadora que realizasse as seguintes operações, guiadas por um conjunto de opções exibidas em uma tela, conforme a figura a seguir:

```
CALCULADORA

0. SOMA
1. SUBTRAÇÃO
2. MULTIPLICAÇÃO
3.DIVISÃO
4. SENO
5. COSSENO
6. TANGENTE
7. MÓDULO
8. POTENCIAÇÃO
9. EXPONENCIAL
10. FATORIAL
11. LOGARITMO NEPERIANO
12. LOGARITMO NA BASE DEZ

Entre com uma opção:
```

Ao ser digitada a opção, deve-se realizar a operação que foi determinada, entretanto, devendo obedecer às condições apresentadas a seguir:

a) Ao ser digitada a opção, deve-se certificar de que esta é válida, ou seja, pertence ao intervalo de 0 a 12.

b) Para a soma, subtração, multiplicação e divisão, dois números serão digitados para a realização da operação. Deve-se verificar ainda que, na divisão, o denominador não poderá ser zero.

c) No cálculo do seno, cosseno e tangente, o valor que será digitado estará em graus, no entanto, para a realização da operação, o ângulo deverá ser transformado para radianos. Deve-se certificar de que, no cálculo da tangente, o argumento da função não poderá ser múltiplo de 90°.

d) No cálculo da potenciação, o programa deverá tratar o caso de se utilizar números negativos na base.

e) Na opção fatorial, o cálculo só pode ser realizado para números inteiros, maiores ou iguais a zero.

f) No cálculo do logaritmo neperiano e na base dez, deve-se verificar se o argumento é um valor maior que zero.

g) Após o cálculo de cada função da calculadora, o programa emitirá uma mensagem, perguntando se o usuário deseja continuar com outro cálculo ou não. Em caso afirmativo, o menu principal será reapresentado e a nova opção escolhida; senão, o programa será encerrado.

6.5. Escreva uma função denominada $InverteCadeia$, que, dada uma cadeia de caracteres S, vai retornar o inverso dessa cadeia. Exemplo: se S='banana', então $InverteCadeia(S)$='ananab'. Dica: utilize as funções de manipulação de caracteres vistas no Capítulo 3.

6.6. Utilizando a função anterior, escreva um fluxograma que leia uma cadeia de caracteres e exiba uma mensagem, dizendo se ela é ou não um palíndromo. Lembrando: uma cadeia de caracteres é um palíndromo se possuir o mesmo significado se for lida da esquerda para a direita ou vice-versa.

6.7. Escreva um fluxograma para a função média aritmética de um vetor de tamanho N. A média aritmética é assim definida:

$$\overline{x} = \frac{\sum_{i=1}^{n} x_i}{n}$$

6.8. Escreva um fluxograma para a função desvio-padrão de um vetor de tamanho N. O desvio-padrão é, então, definido:

$$\overline{x} = \frac{\sum_{i=1}^{n} x_i}{n}$$

$$DM = \frac{\sum_{i=1}^{n} |x_i - \overline{x}|}{n}$$

6.9. As coordenadas de um vetor (da Geometria Analítica) no espaço R^3 podem ser representadas dentro de um vetor do tipo real de tamanho 3 assim: $V[1] = x$, $V[2] = y$, $V[3] = z$. Escreva fluxogramas que implementem as seguintes operações:

a) procedimento para armazenar três valores reais (x, y, z) dentro de um vetor;

b) procedimento que some dois vetores passados como parâmetros e escreva o resultado em um terceiro vetor;

c) função que retorne o produto escalar entre dois vetores, passados como parâmetros;

d) função que retorne o módulo de um vetor passado como parâmetro.

6.10. Escreva o procedimento $TROCA$ que inverterá o valor de seus dois parâmetros x e y.

6.11. Com o procedimento $TROCA$ anteriormente definido, escreva o fluxograma que vai representar o procedimento de ordenação crescente de um vetor de tamanho N pelo "Método das Trocas", descrito no Algoritmo 6.1.

A seguir, verifique a correção do algoritmo, simulando o fluxograma para os seguintes vetores:

- vetor $A = 4$, com $N = 1$;
- vetor $B = 3, 7, -1$, com $N = 3$;
- vetor $C = 0, -1, 9, 3, -1$, com $N = 5$.

Algoritmo 6.1 Algoritmo de ordenação por trocas.

Início
1. $i \leftarrow i$
2. **Enquanto** $i <= N - 1$ **Faça**
3. $j \leftarrow N$
4. **Enquanto** $j > i$ **Faça**
5. **Se** $VETOR[j-1] > VETOR[j]$ **Então**
6. $TROCA(VETOR[j], VETOR[j-1])$
7. **Fim Se**
8. $j \leftarrow j - 1$
9. **Fim Enquanto**
10. $i \leftarrow i + 1$
11. **Fim Enquanto**

Fim

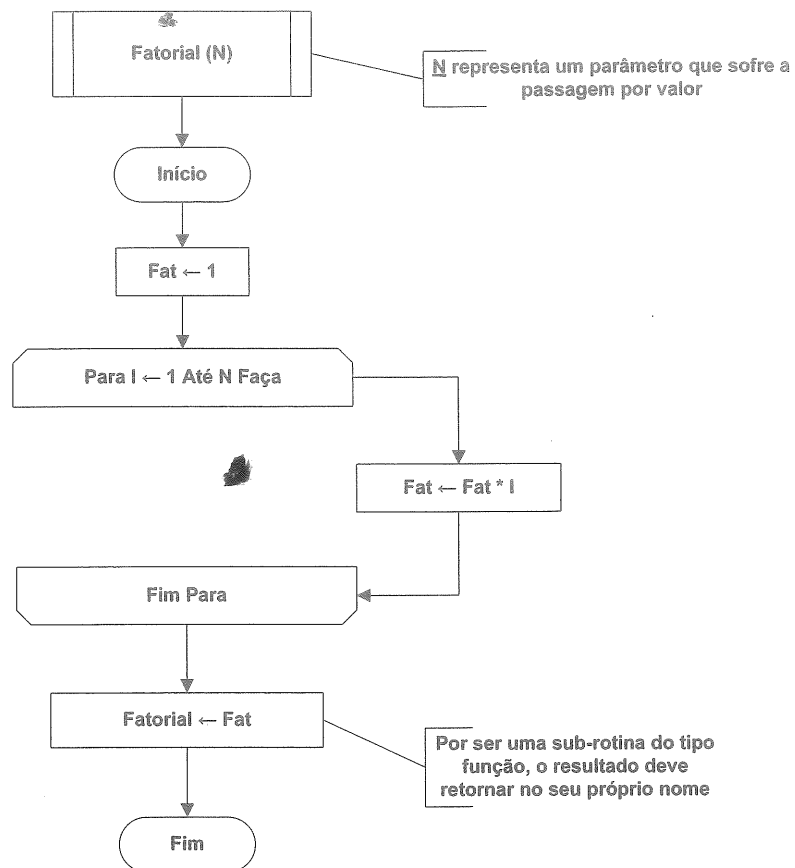
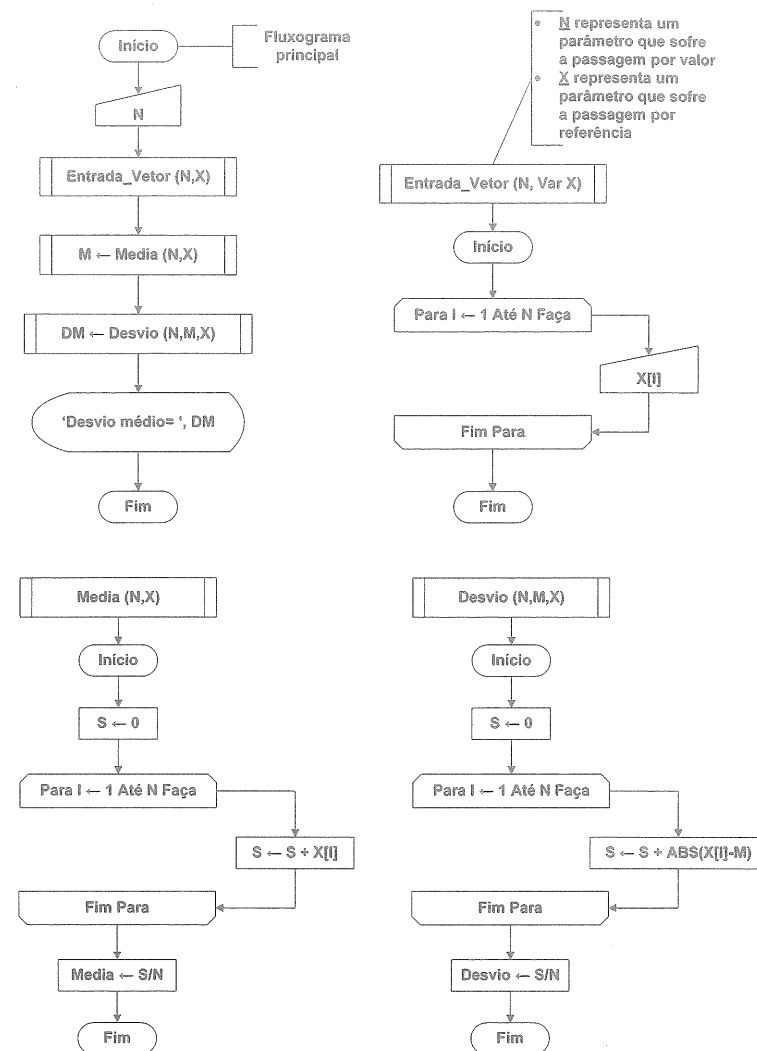
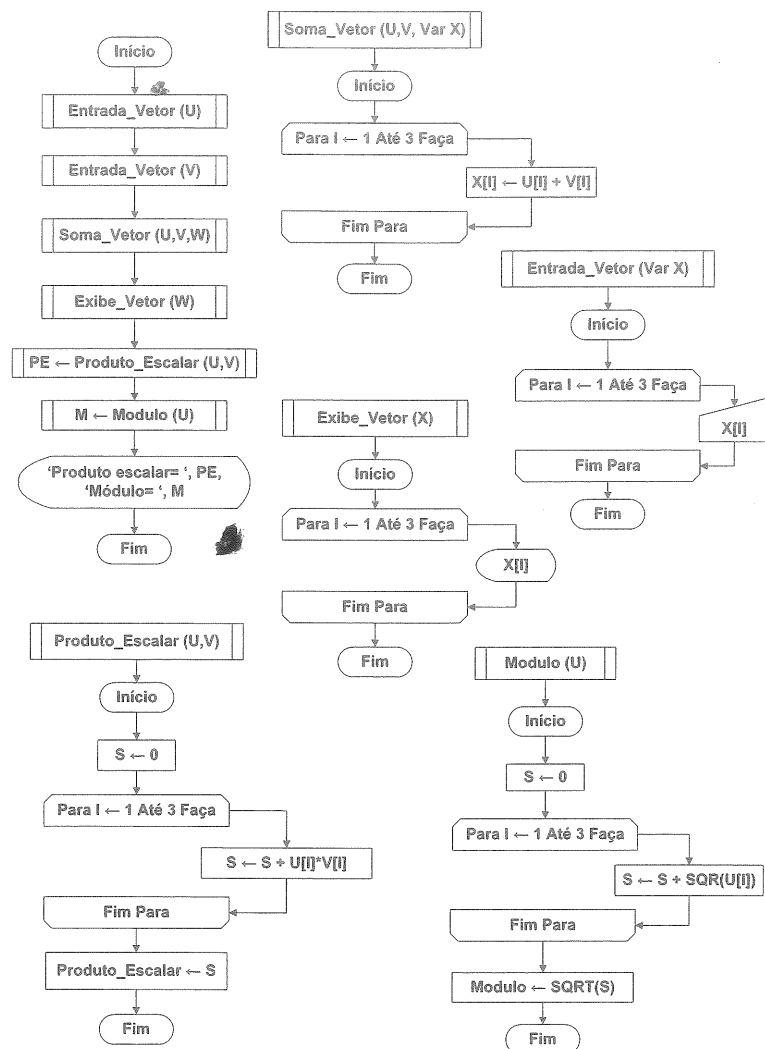
6.12. Construa um fluxograma que leia dois números inteiros (A e B), calculando as seguintes expressões: $X = A!$, $Y = 2A!$ e $Z = (2A)!$. O cálculo do fatorial deve estar escrito em uma sub-rotina.

6.13. Seja a função abaixo:

$$F(X) = X^3 + 3X^2$$

Elabore um fluxograma que faça a leitura de três valores (A, H e N) e a seguir calcule a expressão $P = F(A) + F(A+H) + F(A+2H) + ... + F(A+NH)$, definida pela aplicação de uma função.

6.4 Exercícios resolvidos

6.1.

6.8.

6.9.

Apêndice A

Pequeno Histórico da Computação

A.1 Linha do tempo

1642-44 Blaise Pascal faz sua máquina de calcular.

1790-1801 Joseph-Marie Jacquard desenvolve o mecanismo de cartões perfurados para a confecção de padrões em tecidos feitos em teares mecânicos, uma das primeiras máquinas "programáveis".

1822 Charles Babbage começa a projetar e desenvolver sua "máquina de diferenças".

1833 Babbage usa a ideia dos cartões perfurados para elaborar um dispositivo mecânico programável: a "máquina analítica". Seu filho Henry continuaria a construí-la, mas ela não seria terminada.

1854 George Boole publica trabalhos em que tenta descobrir leis algébricas para o pensamento. Seu trabalho será a base lógica dos cálculos nos futuros computadores, fundamentando a álgebra booleana.

1870 (por volta de) William Thomson, conhecido como Lord Kelvin (1824-1909), cria uma máquina analógica para prever as marés, a qual dará origem aos primeiros computadores analógicos.

1890 O censo dos Estados Unidos usa uma tabuladora desenvolvida por Hermann Hollerith no processamento dos resultados, com uso de cartões perfurados, no padrão do tear de Jacquard.

1896 Hollerith funda a *Computing-Tabulating-Recording*, antecessora da IBM.

1924 Fundação da IBM (sigla para International Business Machines), fabricando perfuradoras de cartões.

1934 O alemão Konrad Zuse começa a fazer uma máquina eletromecânica programável.

1935 O norte-americano John Vincent Atanasoff começa a fazer uma máquina eletrônica para resolver sistemas específicos. A máquina usa leitora e perfuradora de cartões.

1939 Primeira calculadora eletromecânica feita à base de relés, desenvolvida nos Laboratórios Telefônicos Bell por George Stiblitz.

1941 Konrad Zuse conclui um computador eletromecânico, o Z3, que é depois destruído em Berlim, na Segunda Guerra Mundial.

1942 John Vincent Atanasoff conclui sua máquina, o ABC, capaz de resolver problemas reduzidos a sistemas de 30 equações.

1944 Howard Aiken e sua equipe da Universidade de Harvard e da IBM concluem o Mark-1, computador eletromecânico que funcionava com relés e era programado por fita de papel.

1941-45 O matemático britânico Alan Mathison Turing lidera o desenvolvimento de uma máquina para decifrar o código secreto das máquinas alemãs Enigma. Em Bletchey Park (Londres), a equipe constrói o Colossus, que ficou pronto em 1943 e foi o primeiro computador digital específico para quebrar códigos.

1946 É revelado ao público o primeiro computador totalmente eletrônico e digital de aplicação geral: o Eniac, desenvolvido pela Escola Moore da Universidade da Pensilvânia e pelo Laboratório de Pesquisas Balísticas do Exército dos Estados Unidos.

1947 O primeiro transistor é desenvolvido para substituir as válvulas. A partir da década de 50, a transistorização será o primeiro marco revolucionário no aumento da velocidade das máquinas. Em 1948 é inventado um corretor de erros para computadores.

1949 Na Universidade de Cambridge, a equipe de Maurice Wilkes conclui o primeiro computador eletrônico digital que armazenava o próprio programa, o Edsac (Computador Automático de Armazenamento Eletrônico por Atraso).

1951 Os criadores do Eniac, John Presper Eckert Jr. e John Mauchly, lançam o primeiro computador que armazenava programas e estava disponível comercialmente, o chamado Univac-1 (sigla para Computador Automático Universal).

1951 Conclusão do Edvac (Computador Eletrônico de Variável Discreta), idealizado pela equipe do Eniac. Além de seguir a arquitetura proposta por John von Neumann, o Edvac contava com um dispositivo chamado "linhas de atraso", elaborado por J. Presper Eckert Jr., que multiplicava a capacidade de armazenamento e reduzia o tamanho da memória.

1952 Conclusão do computador do Instituto de Estudos Avançados da Universidade de Princeton, sob supervisão de John von Neumann.

1953 Primeiro computador eletrônico digital da IBM, o IBM-701.

1957 Aparecimento do Fortran (FORmula TRANslator), linguagem elaborada dentro da IBM, conhecida como a primeira linguagem de alto nível. O Fortran tornou mais fácil a atividade de programar os computadores e pode ser considerado o primeiro passo rumo a sistemas mais "amigáveis".

1957 Primeiro leitor de disquetes (*disk drive*) comercial com cabeça para leitura e gravação, o IBM-305.

1959 Primeira máquina que ficou conhecida como minicomputador, o PDP-1 da Digital Equipment Corporation.

1960 Primeira linguagem voltada para uso em programação comercial, o Cobol (sigla para Linguagem Orientada para Negócios Comuns – Common Business Oriented Language).

1961 Primeiro circuito integrado disponível comercialmente, desenvolvido na Fairchild Corporation durante um período de três anos.

1963 Primeiro uso confiável de terminais com monitores de vídeo, em um minicomputador PDP-1.

1964 Primeiro dispositivo de entrada conhecido como "mouse", desenvolvido por Douglas Engelbart.

1969 Ken Thompson e Dennis Ritchie desenvolvem, nos Laboratórios Bell, o sistema operacional Unix, o primeiro sistema operacional geral que poderia ser aplicado

em qualquer máquina. O sistema, de características gerais, também daria origem à linguagem de programação C.

1969 O exército norte-americano conecta máquinas da Arpanet, rede que originaria a Internet.

1971 Primeiro microprocessador (*chip*) disponível comercialmente, o Intel 4004.

1973 Primeiro computador pessoal completo com monitor, o Alto, desenvolvido pela Xerox.

1975 Primeiro computador pessoal produzido para consumo em massa, o Altair 8800.

1975 Primeiro computador pessoal da IBM, IBM 5100.

1976 Apple II se torna o computador pessoal mais bem-sucedido comercialmente.

1979 Criação da planilha eletrônica Visicalc para o sistema operacional CPM em micros Apple II.

1981 A IBM lança seu computador pessoal IBM-PC, o primeiro vendido com sucesso. O sistema operacional do computador é o MS-DOS, desenvolvido em parceria com a Microsoft.

1983 A Borland lança o compilador Turbo Pascal, primeira linguagem com um ambiente integrado de desenvolvimento (*IDE – Integrated Development Enviroment*), em que o programa pode ser editado, compilado, "linkado" e testado sem necessidade de sair do ambiente. Até aqui, os programas compilados precisavam ser editados em um editor de textos, salvos; encerrado o editor, passado o arquivo texto ao compilador, o qual gerava o código objeto. Após essas operações, era destinado ao "linker", programa que faz a ligação de bibliotecas de funções, para, então, ser criado o programa executável. Tudo isso se o compilador não apresentasse nenhum erro. Possui versões para o Apple em CPM e para o IBM-PC com DOS.

1984 A Apple lança o computador pessoal Macintosh, com sistema operacional baseado em figuras para acionar comandos e facilita o diálogo com o usuário.

1984 Lançamento pela Borland da versão 2.0 do compilador Turbo Pascal.

1985 Lançamento pela Borland da versão 3.0 do compilador Turbo Pascal.

1987 Lançamento pela Borland da versão 4.0 do compilador Turbo Pascal. Introduz o conceito de *units* (bibliotecas de sub-rotinas).

1989 Lançamento pela Borland da versão 5 do compilador Turbo Pascal. Introduz o conceito de objetos.

1990 A Microsoft lança versão 3.0 do programa "Windows", baseado no sistema do Macintosh, para ser usado nos computadores que utilizam o MS-DOS.

1990 Lançamento pela Borland da versão 6.0 do compilador Turbo Pascal, juntamente com o Turbo Vision, uma biblioteca orientada a objetos.

1992 Lançamento pela Borland da versão 7.0 do compilador Turbo Pascal, para programação DOS, DOS protegido e Windows, com base na tecnologia Object Windows.

1993 Lançamento pela Borland da primeira versão do compilador Delphi, baseado em tecnologia RAD (*Rapid Application Development*), facilitando, e muito, a programação para o Windows. Esse ambiente (o Delphi) é totalmente baseado no Object Pascal, fusão das tecnologias lançadas desde 1987.

1995 A Microsoft faz lançamento mundial em agosto do sistema operacional "Windows 95".

1995 Lançamento pela Borland da versão 2.0 do Delphi. Adapta o compilador para sistema 32 bits.

1996 Lançamento pela Borland da versão do Delphi 3.0. Essa versão introduz novos componentes.

1998 A Microsoft lança o "Windows 98".

1998 Lançamento pela Borland da 4.0 do Delphi. Introduz novos componentes, principalmente para a Internet, compatibilizando suporte a Corba e DCOM, tecnologias de dados distribuídos.

1999 Lançamento pela Borland da versão 5.0 do Delphi para Windows.

2000 Anúncio de lançamento pela Borland da versão 6.0 do Delphi, agora com suporte ao sistema operacional Linux.

17/02/00 Lançamento oficial do Windows 2000.

Apêndice B

A Norma ISO 5807/1985

A norma ISO 5807/1985 classifica os símbolos de acordo com sua utilização, em:

1. **Símbolos básicos**: utilizados quando a natureza precisa ou forma, por exemplo, do processo ou da mídia dos dados, é desconhecida ou, ainda, não necessário discriminar a mídia atual.

2. **Símbolos específicos**: utilizados quando a natureza precisa do processo ou da mídia dos dados é conhecida ou deve ser explicitada.

A partir dessa classificação, a norma explicita sua aplicação a:

1. ***Data Flowcharts*** (fluxogramas de dados): representação do caminho dos dados na solução do problema e definição dos passos de processamento.

2. ***Program Flowcharts*** (fluxogramas de programas): representação da sequência de operações em um programa.

3. ***System Flowcharts*** (fluxogramas de sistemas): representação do controle das operações e do fluxo dos dados de um sistema.

4. ***Program Network Chart*** (diagrama de programa em rede): representação dos caminhos de ativação dos programas e das iterações com os dados relacionados.

5. ***System Resources Chart*** (diagrama de recursos do sistema): representação das configurações das unidades de dados e unidades de processos adaptáveis para a solução de um problema ou de um conjunto de problemas.

B.1 Os símbolos

Os símbolos são agrupados em quatro categorias: dados, processos, linhas e especiais. Dentro de cada categoria, são divididos em básicos e específicos, conforme mencionado anteriormente. Observe que, apesar de a norma ser de 1985, ela está atualizada em termos da tecnologia hoje em uso.

A aplicação é genérica, buscando o aproveitamento do enorme potencial de comunicação dos símbolos visuais em substituição aos longos textos descritivos. Essa é a principal razão do uso de fluxogramas na engenharia. A noção das etapas de processamento, do que consistem as entradas e saídas e todo o percurso de material, dados, produtos e o que mais for necessário avaliar, está claramente mostrada e pode ser facilmente visualizada. Além disso, a aplicabilidade assume o âmbito geral, não se restringindo apenas ao meio de visualização de lógica de programação. A adaptação dos símbolos às atividades industriais e comerciais é evidente, razão pela qual, coerentemente, é adotada na representação de processos de qualificação ISO 9000.

A estrutura de símbolos básicos e símbolos específicos permite a análise e a representação do mesmo sistema de formas distintas, possibilitando, assim, diferentes níveis da análise. É também possível dessa maneira se trabalhar formas genéricas que se adaptem a novas tecnologias. O ensino/aprendizado da arte da programação de computadores sai fortalecido por uma representação que pode ser adaptada a qualquer linguagem de programação.

B.1.1 Símbolos relativos a dados

Tabela B.1 Tabela de símbolos da ISO 5807 relativos a dados.

Símbolo	*Nome* *Classificação*	*Utilidade*
	Dados *Básico*	Representar os dados, tanto de entrada como de saída, qualquer que seja a mídia utilizada.
	Dados armazenados *Básico*	Representar os dados armazenados de forma ajustável para processamento com mídia não específica.
	Armazenamento interno *Específico*	Representar os dados armazenados internamente.
	Acesso a armazenamento sequencial *Específico*	Representar os dados armazenados, de acesso somente sequencial, cuja mídia seja, por exemplo, fita magnética, fita cassete, *tape cartdrige* etc.
	Acesso a armazenamento direto *Específico*	Representar os dados acessíveis de forma direta ("aleatória"), cuja mídia é, por exemplo, disco magnético, "winchester", disco flexível etc.
	Documento *Específico*	Representar os dados legíveis por seres humanos, cuja mídia seja, por exemplo, saída impressa, um documento "escaneado", microfilme, formulário impresso de dados etc.
	Entrada manual *Específico*	Representar os dados, de qualquer tipo de mídia, que sejam inseridos manualmente em tempo de processamento, por exemplo, teclado *on-line*, mouse, chaveamento, caneta óptica *light pen*, leitor de código de barras etc.
	Cartão *Específico*	Representar os dados cuja mídia seja cartão perfurado, magnéticos, de marcas sensíveis etc.
	Fita perfurada *Específico*	Representar os dados cuja mídia seja fita perfurada.
	Exibição *Específico*	Representar os dados cuja mídia seja de qualquer tipo na qual a informação seja mostrada para uso humano, tais como monitores de vídeo, indicadores *on-line*, mostradores etc.

B.1.2 Símbolos relativos a processos

Os símbolos de processos são apresentados a seguir. Dentre os símbolos de processos, surgem modificações em relação às normas de 1971 da ANSI. Deve-se notar aqui uma forma especial de representar os laços de repetição, tendo-se em vista as estruturas do tipo laço controlado "FOR ... TO... DO" das linguagens de programação. É também possibilitada uma representação para a estrutura "CASE... OF" do Pascal ou a "SWITCH" da linguagem C, a partir da estrutura condicional normal.

Essas modificações são as grandes responsáveis pela adaptação às linguagens estruturadas. A existência de símbolo para a representação de módulos e de sub-rotinas permite perfeitamente a criação e representação de programas com a adoção da metodologia "TOP-DOWN" ou "BOTTOM-UP", derrubando por terra a crença contrária.

Tabela B.2 Tabela de símbolos da ISO 5807 relativos a processos.

Símbolo	*Nome* *Classificação*	*Utilidade*
	Processo *Básico*	Representar qualquer tipo de processo, processamento de função, por exemplo, executando uma operação definida ou grupo de operações, resultando na mudança de valor, forma ou localização de uma informação, ou determinação de uma, entre várias direções de fluxo, a ser seguida.
	Processo pré-especificado *Específico*	Representar um processo nomeado, consistindo em um ou mais passos de programa que são especificados em outro local, como, por exemplo, uma sub-rotina ou módulo de programa.
	Operação manual *Específico*	Representar uma operação manual, ou seja, qualquer processo executado por um ser humano.
	Preparação *Específico*	Representar modificação de uma instrução ou grupo de instruções, de forma a afetar a atividade subsequente, por exemplo, configurar uma chave, alavanca, modificar o indexador de um registro ou preparar uma rotina.
	Decisão *Específico*	Representar uma decisão ou um desvio tendo uma entrada, porém pode ter uma série de saídas alternativas, uma única das quais deverá ser ativada como consequência da avaliação das condições internas ao símbolo. O resultado apropriado de cada saída deverá ser escrito adjacente à linha, representando o caminho respectivo.
	Modo paralelo *Específico*	Representar processamento paralelo ou a sincronização de duas ou mais operações paralelas.
	Limitador de laço repetitivo *Específico*	Início e final de laço (*loop*) controlado. Deve existir em conjunto com o símbolo que mostra o final do laço. As condições de inicialização, incremento, terminação etc. devem aparecer dentro do símbolo respectivo, de acordo com a posição da operação de teste.

B.1.3 Símbolos de linhas

Tabela B.3 Tabela de símbolos da ISO 5807 relativos a processos.

Símbolo	*Nome* *Classificação*	*Utilidade*
	Linha básica *Básico*	Representar o fluxo dos dados ou controles. Podem ser utilizadas pontas de seta, sólidas ou abertas, na extremidade para indicar a direção do fluxo onde é necessário ou para enfatizá-lo e facilitar a legibilidade.
	Transferência de controle *Específico*	Representar a transferência de controle de um processo para outro, algumas vezes com oportunidade de retorno direto para ativação de processos, após estes completarem suas ações. O tipo de transferência pode ser nomeado no símbolo, por exemplo, chamada (*call*), evento etc.
	Link de comunicação *Específico*	Representar a transferência de dados por um "link" de telecomunicações.
--------	***Linha tracejada*** *Específico*	Representar um relacionamento alternativo entre dois ou mais símbolos. Também é utilizado para delimitar uma área de anotações.

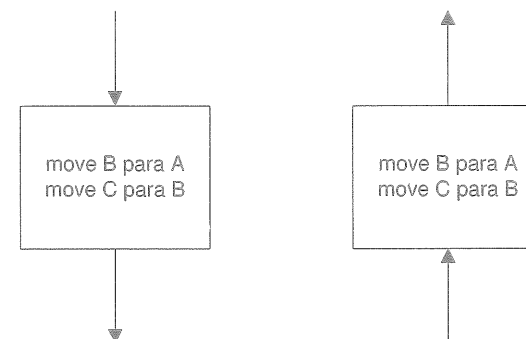
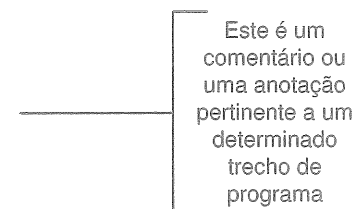
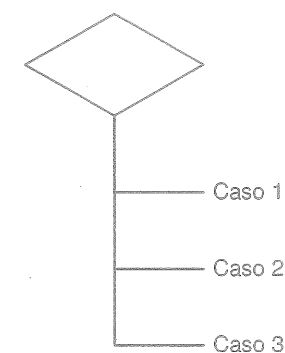
B.1.4 Símbolos especiais

Tabela B.4 Tabela de símbolos especiais da ISO 5807.

Símbolo	*Nome*	*Utilidade*
○	*Conector*	Representar a saída para, ou a entrada de outra parte do mesmo fluxograma, usada para quebrar uma linha e continuá-la em outra parte. Os símbolos de conexão devem possuir o mesmo identificador (único) interno.
	Terminador	Representar a saída para, ou a entrada do ambiente externo, por exemplo início ou final de programa, uso externo e origem ou destino de dados etc.
	Anotação	Representar o adicionamento de comentários para esclarecimento ou explanação de observações. Devem ser utilizadas linhas tracejadas ligando ou cercando o(s) símbolo(s) respectivo(s), próximo(s) e ao seu redor.
—...—	*Elipse*	Representar a omissão de um ou mais símbolos em que nem o tipo, nem o número de símbolos estão definidos. Esse símbolo é utilizado entre linhas e se aplica ao caso de diagramas, mostrando soluções gerais com um número aberto de repetições.

Convenciona-se, ainda, que os símbolos tenham por finalidade representar e identificar graficamente a função à qual eles representam, independentemente do texto interno. Aconselha-se que o espaçamento entre os símbolos seja o mais uniforme possível, assim como suas dimensões, e que se evite uso de linhas longas e alterações dos ângulos dos formatos. Os símbolos podem ser desenhados segundo qualquer orientação, porém, preferencialmente na horizontal.

B.1.5 Textos internos

Os textos internos devem ser os menores possíveis, devendo ser lidos da esquerda para a direita, de cima para baixo, independentemente da orientação das linhas de fluxo (veja figura a seguir). Quando a quantidade de texto for muito grande para ser comportada pelo símbolo, um símbolo de anotação pode ser utilizado para completá-lo. Quando a anotação atrapalha o fluxo do diagrama, o texto pode ser colocado em um formulário separado com referência cruzada no diagrama.

A norma convenciona, ainda, a possibilidade de se utilizar identificadores e descritores dos símbolos para serem referenciados em outras partes da documentação, detalhamento de processos ou de dados, convenções de linhas de conexão, do uso dos conectores e representação de múltiplas saídas. Veja os casos a seguir:

Apêndice C

Operadores e Funções Predefinidas

A norma ISO 5807/1985 nada especifica sobre a notação a ser utilizada nos símbolos. Os símbolos adotados são baseados no livro *The Art of Computing Programming – Fundamental Algorithms*, de Donald Knuth.

C.1 Operadores matemáticos

Tabela C.1 Operadores matemáticos.

Símbolo	Significado
$\leftarrow$	Atribuição de valores
$<$	Operador relacional MENOR QUE
$>$	Operador relacional MAIOR QUE
$<=$	Operador relacional MENOR OU IGUAL A
$>=$	Operador relacional MAIOR OU IGUAL A
$=$	Operador relacional IGUAL A
AND	Conector de INTERSEÇÃO
OR	Conector de UNIÃO
NOT	Conector de NEGAÇÃO
$true$	Valor booleano verdadeiro
$false$	Valor booleano falso
$+, -, *, /$	Operadores aritméticos de adição, subtração, multiplicação e divisão
[ ]	Marcação de índices de variáveis indexadas
( )	Prioridade aritmética na expressão
DIV	Operador de divisão inteira
MOD	Operador de resto da divisão inteira

C.2 Funções predefinidas

Consideram-se válidas também as seguintes funções e procedimentos predefinidos, comuns a várias linguagens de programação de alto nível:

Tabela C.2 Funções predefinidas.

Função	Utilidade
$ln(x)$	Retorna o valor do logaritmo neperiano de x
$exp(x)$	Retorna o valor de e^x
$int(x)$	Retorna a parte inteira de x (real) como número real
$frac(x)$	Retorna a parte decimal de x (real) como número real
$trunc(x)$	Retorna a parte inteira de x (real) como número inteiro
$round(x)$	Arredonda para o próximo inteiro o valor de x real
$sqr(x)$	Retorna o quadrado de x (real)
$sqrt(x)$	Retorna o valor da raiz quadrada de x (real)
$sin(x)$	Retorna o valor do seno de x, x medido em radianos
$cos(x)$	Retorna o valor do cosseno de x, x medido em radianos
$arctan(x)$	Retorna o valor do arco, em radianos, cuja tangente vale x
$concat(s1, s2)$	Retorna uma cadeia de caracteres unindo $s2$ ao final de $s1$
$length(s)$	Retorna o número de caracteres que compõe a cadeia s
$pos(s1, s2)$	Retorna o valor da posição na qual começa a cadeia $s1$ na cadeia $s2$
$copy(s, p, n)$	Retorna nova cadeia de caracteres com n elementos de s a partir da posição p
$insert(s1, s2, p)$	Retorna nova cadeia $s2$, com $s1$ inserida a partir da posição p

As expressões que permitem calcular os valores que não fazem parte dos predefinidos são obtidas com o uso das funções predefinidas. Alguns exemplos são descritos na Tabela C.3.

Tabela C.3 Expressões derivadas de funções predefinidas.

Função	Expressão analítica	Forma linear
Tangente de x	$tan(x) \leftarrow \frac{sin(x)}{cos(x)}$	$tan \leftarrow sin(x)/cos(x)$
Cotangente de x	$cotan(x) \leftarrow \frac{cos(x)}{sin(x)}$	$cotan \leftarrow cos(x)/sin(x)$
Secante de x	$sec(x) \leftarrow \frac{1}{cos(x)}$	$sec \leftarrow 1/cos(x)$
Cossecante de x	$cosec(x) \leftarrow \frac{1}{sin(x)}$	$cosec \leftarrow 1/sin(x)$
Arco cujo seno vale x	$arcsin(x) \leftarrow arctan(\frac{x}{\sqrt{1-x^2}})$	$arcsin \leftarrow arctan(x/sqrt(1-sqr(x)))$
Arco cujo cosseno vale x	$arccos(x) \leftarrow arctan(\frac{\sqrt{1-x^2}}{x})$	$arccos \leftarrow arctan(sqrt(1-sqr(x))/x)$
A elevado a B	$Z \leftarrow A^B = e^{B \cdot ln(A)}$	$Z \leftarrow exp(B * ln(A))$

A adoção dessa notação para a área de informática deve satisfazer a maior parte dos usuários de computadores, uma vez que é bastante próxima da maioria das linguagens mais atuais.

Referências Bibliográficas

BALL, R.; COXETER, H. S. M. *Mathematical recreations and essays*. Nova York: Dover Publications, Inc., 1987.

BÖHM, C.; JACOPINI, G. Flow diagrams, turing machines and languages with only two formation rules. *Comunications of the ACM*, v. 9, n. 5, p. 366–371, maio 1966.

BOYER, C. B. *História da matemática*. São Paulo: Edgard Blücher, 1974.

CLESSA, J. J. *Math and logic puzzles for PC enthusiasts*. Nova York: Dover Publications, Inc., 1996.

DAVIS, H. T. *Tópicos de história da matemática para uso em sala de aula: computação*. São Paulo: Atual, 1995.

GONICK, L. *Introdução ilustrada à computação*. São Paulo: Harbra, 1984.

HAREL, D. *Algorithmics: the spirit of computing*. Workingham, Inglaterra: Addison-Wesley Publishing Company, 1987.

HAYES, J. P. *Computer architecture and organization*. 3. ed. Nova York: McGraw-Hill, 1998. (Computer Science Series).

KNUTH, D. E. *Fundamental algorithms*. Reading, Massachusetts: Addison-Wesley, 1972. (The Art of Computer Programming, v. 1).

MICHALEWICZ, Z.; FOGEL, D. B. *How to solve it: modern heuristics*. Berlim: Springer-Verlag, 2000.

PFLEEGER, S. L. *Software engineering: theory and practice*. 2. ed. Upper Saddle River, New Jersey: Prentice Hall Inc., 2001.

POHL, I.; SHAW, A. C. *The nature of computation: an introduction to computer science*. Rockville, Maryland: Computer Science Press, 1981. (Computer Software Engineering Series).

POLYA, G. *How to solve it: a new aspect of mathematical method*. 2. ed. Princeton, New Jersey: Princeton University Press, 1985.

TANENBAUM, A. S. *Structured computer organization*. 4. ed. Upper Saddle River, New Jersey: Prentice Hall, 1999.

TREMBLAY, J.-P.; BUNT, R. B. *An introduction to computer science: an algorithmic approach*. Nova York: McGraw-Hill, 1979.

WEINBERG, G. M. *The psychology of computer programming*. Nova York: Van Nostrand Reinhold, 1971.

WIRTH, N. *Programação sistemática em Pascal*. 2. ed. Rio de Janeiro: Campus, 1982.

Lista de Crédito das Figuras

Figs. 1.1, 1.2 e 1.3: Eduardo Borges

Fig. 2.2: ©2011 Photos.com, uma divisão da Getty Images. Todos os direitos reservados.

Fig. 2.4: Apesar de todos os esforços, não encontramos o responsávelpela foto.

Fig. 2.6: J.-L. CHARMET/SCIENCE PHOTO LIBRARY/SPL DC/Latinstock

Fig. 2.7: SCIENCE PHOTO LIBRARY/SPL DC/Latinstock

Fig. 2.8: http://de.wikipedia.org/w/index.php?title=Datei:Babbages_difference_engine_1832.jpg&filetimestamp=20080514050704

Fig. 2.9: SCIENCE PHOTO LIBRARY/SPL DC/Latinstock

Fig. 2.10: Stiftung Deutsches Technikmuseum Berlin

Fig. 2.11: Apesar de todos os esforços, não encontramos o responsável pela foto.

Fig. 2.12: IBM

Fig. 2.13: Apesar de todos os esforços, não encontramos o responsável pela foto.

Fig. 2.14: LOS ALAMOS NATIONAL LABORATORY/SCIENCE PHOTO LIBRARY/SPL DC /Latinstock

Fig.2.15: http:/pt.wikipedia.org/wiki/Ficheiro.Edvac.jpg

Fig. 2.16: Alan Rocha dos bancos Shelby e Leon Levy Archives Center, Instituto de Estudos Avançados, em Princeton, NJ, EUA

Fig. 2.18: ©Bettmann/CORBIS/Corbis (DC)/Latinstock

Fig. 2.19: IBM

Fig. 2.20: IBM

Fig. 2.21: UNIVERSITY CORPORATION FOR ATMOSPHERIC RESEARCH/SCIENCE PHOTO LIBRARY/SPL DC Latinstock

Fig. 2.23: IBM

Fig. 2.23: IBM

Sobre os Autores

Marco Antonio Furlan de Souza é Engenheiro Eletricista pela Faculdade de Engenharia Industrial (FEI) e mestre em Engenharia Elétrica pela Escola Politécnica da Universidade de São Paulo. É professor das disciplinas Engenharia de Software, Sistemas de Informação, Programação Orientada a Objetos com Java, Métodos de Otimização, Compiladores e Computação I e II da Escola de Engenharia Mauá.

Marcelo Marques Gomes é Engenheiro Eletricista pela Escola de Engenharia Mauá e mestrando em engenharia elétrica pela Universidade Estadual de Campinas. É professor das disciplinas Métodos Numéricos e Visão Computacional da Escola de Engenharia Mauá.

Marcio Vieira Soares é Engenheiro Naval pela Escola Politécnica da Universidade de São Paulo. É professor das disciplinas Algoritmos e Programação, Complementos de Computação e Computação I da Escola de Engenharia Mauá.

Ricardo Concilio é Engenheiro Eletricista pela Escola de Engenharia Mauá e mestre em Engenharia Elétrica pela Universidade Estadual de Campinas. Atualmente é doutorando em Engenharia Elétrica pela mesma instituição. É professor da disciplina Algoritmos e Programação e Métodos Numéricos da Escola de Engenharia Mauá.